Interactive Authentication

Deepak Maram
Cornell Tech and *IC3*
sm2686@cornell.edu

Mahimna Kelkar
Cornell Tech and *IC3*
mahimna@cs.cornell.edu

Ittay Eyal
Technion and *IC3*
ittay@technion.ac.il

***Abstract*—Access to digital services like online banking and social networks and control of assets like cryptocurrencies relies on principals maintaining credentials for authentication. But both individuals and organizations struggle to manage their credentials, resulting in loss of assets and identity theft. Multi-factor authentication improves security, but its analysis and design are mostly limited to *one-shot mechanisms*, which decide immediately.**

In this work, we study mechanisms with back-and-forth *interaction* with the principals. For example, a user receives a mobile alert about a sending money from her bank account and is given a time period to abort before the operation is executed.

We formally define *the authentication problem*, where an *authentication mechanism* interacts with a user and an attacker and tries to identify the user. A mechanism's success depends on the scenario – the state of the different credentials (safe, lost, leaked, or stolen). The *profile* of a mechanism is the set of all scenarios in which it succeeds. Thus, we have a partial order on mechanisms, defined by the subset relation on their profiles. We find an upper bound on the profile size. For any number of credentials n, we find three types of mechanisms that are *maximally secure*, meeting the above bound. We show these are all the unique maximal mechanisms for $n \leq 3$.

We show the practical applicability of our framework by modeling existing mechanisms, both from academic literature and deployed in widely-used systems. We make concrete improvement proposals including an implementation of a practical, maximally-secure cryptocurrency wallet.

I. Introduction

Authentication plays a critical role in safeguarding online services and digital assets. An *authentication mechanism* binds an owner (physical identity) to a digital identity. It relies on the owner's (exclusive) access to *credentials* like a password, a one-time PIN (OTP), or a cryptographic signing key. But credential management is challenging: for example, identity theft in the US is rampant [1]; an estimated 20% of Bitcoin have disappeared as a result of key loss [2]; and $600M in cryptocurrency were stolen recently due to a single company's key mismanagement [3].

Some practical mechanisms employ *interaction*. For example, in the Argent cryptocurrency wallet [4], any 1-out-of-3 credentials can initiate a change of credentials, but the change only happens after two days, during which the operation can be cancelled by showing any 2-out-of-3 credentials. In online banking, some banks (e.g., [5]) introduce an artificial delay period before transferring funds; they notify the client and allow her to abort an erroneous transaction during this period. US government agencies (e.g., [6]) send a mail notice about reversible account activity, allowing victims to revert in case of identity theft attempts [1].

But, to the best of our knowledge, interactive authentication has never been formally studied and its importance is therefore under-appreciated. For example, multi-factor authentication is typically defined as requiring multiple credentials [7], i.e., combine credentials using a conjunction (AND) operator, without taking advantage of interactivity. Similarly, multi-sig or threshold mechanisms (t-out-of-n) are popular in the cryptocurrency industry [8], [9], [10], but are non-interactive. Prior works (§II) have focused on proposing frameworks that only model non-interactive mechanisms [11], [12], or consider specific interactive mechanisms [4], [13], [14], [15].

In this work, we formalize the *authentication problem* (§III), consisting of a *mechanism*, a *user*, and an *attacker* that stands for all entities trying to authenticate as the user.

Say that the mechanism uses some n credentials ($n \geq 1$) to identify the user. Each of these credentials can be in one of four states [11]: *safe* (only the user has it), *stolen* (only the attacker has it), *leaked* (both have it) or *lost* (neither has it). The states of all the n credentials together define a *scenario*, with a total of 4^n scenarios possible.

Given a scenario, the parties interactively exchange messages with the mechanism over a *synchronous* channel until it *decides* which of them is the true owner. A mechanism *succeeds* if it is correct irrespective of what the attacker does. Otherwise the mechanism *fails*.

To evaluate mechanisms, we define the *security profile* of a mechanism as the set of all scenarios in which it is successful. For example, with two credentials denoted by c_1 and c_2, let us consider the OR-mechanism where either c_1 or c_2 can be used to authenticate, i.e., $c_1 \vee c_2$. Its profile has three scenarios: (c_1 and c_2 safe), (c_1 safe, c_2 lost) and (c_1 lost, c_2 safe). The mechanism succeeds in the latter two because the user knows enough credentials to authenticate and the attacker does not. Note that the mechanism fails in scenarios where both parties have one credential, e.g., (c_1 stolen, c_2 safe), because we conservatively assume that the adversary controls the order in which messages are delivered to the mechanism.

The security profile thus defines a relation between any two mechanisms M_1 and M_2 using the same number of credentials: M_1 is *better* than M_2 if the profile of M_1 is a superset of M_2. M_1 is *equivalent* to M_2 if they have the same profile up to credential permutation. Otherwise, M_1 and M_2 are incomparable. A mechanism is *maximally secure* or simply *maximal* if no other mechanism is better.

By proving constraints on the profile sets any mechanism could achieve, we bound the profile size (§IV-A). We find

that with n-credentials, no mechanism succeeds in more than $P(n) = \frac{4^n - 2^n}{2}$ scenarios. For example, $P(2) = 6$, that is, any 2-credential mechanism can win in at most 6 scenarios.

The bound is tight as we discover several maximal mechanisms meeting it. All our maximal mechanisms function as follows: Either party can initiate the mechanism, at which point the mechanism starts a timer. Until the time runs out, each party can submit messages to the mechanism, carrying one or more credentials. Finally, the mechanism decides on the winner. We refer to mechanisms (maximal or otherwise) following the above structure as *bounded-delay mechanisms*. They only differ in the way a winner is decided.

We identify three sufficient properties for a bounded-delay mechanism to be maximal (§IV-B): (1) *ID-agnostic*: the mechanism only decides based on messages, not other information; (2) *knowledge-rewarding*: if a party submits a credential set that is the superset of its counterpart, then the mechanism decides on the former; and (3) *transitive-knowledge-rewarding*: submitting additional credentials cannot lead to a worse outcome for any party.

We discover three classes of mechanisms that satisfy these properties (§IV-C). The first two are fairly intuitive: *Majority mechanisms* decide based on who submits the most credentials, with some tie-breaking function. *Priority mechanisms* use a priority vector defined over the n-credentials, and the winner is the party that submits a unique higher priority credential. For example, consider the priority vector $[c_2, c_3, c_1]$: if party 1 (P1) submits $\{c_1, c_2\}$ and party 2 (P2) submits $\{c_2, c_3\}$, then P2 wins because c_3 has a higher priority than c_1.

The third class is similar to the priority mechanisms but with an exception: if only the last two credentials in the priority vector are submitted, then the priority is reversed. Using the same vector from before, if P1 submits $\{c_1\}$ and P2 submits $\{c_3\}$, then P1 wins. This is the only exception and we use the priority rule otherwise, e.g., if P1 submits $\{c_1, c_2\}$ and P2 submits $\{c_2, c_3\}$, then P2 wins like before.

We can now illustrate the advantage of interactivity by going back to the 2-credential setting. Consider the 2-credential priority mechanism defined by the vector $[c_1, c_2]$. Its profile contains (c_1 and c_2 safe); (c_1 safe, c_2 leaked or lost or stolen) because only the user knows the highest-priority credential c_1; and (c_1 leaked or lost, c_2 safe) because c_2 is safe and c_1 is known to either both the parties or to neither. In total, the size of the 2-credential priority mechanism's profile is six, equal to the bound $P(2)$ and three more than the OR-mechanism described above.

The three mechanism classes cover the *complete sets* of n-credential maximal mechanisms for $n \leq 3$, i.e., *any* mechanism *must be either worse than or equivalent* to a mechanism in the three classes. In more detail, the complete set of 2-credential mechanisms has just the priority mechanism introduced before. Whereas the complete set of 3-credential mechanisms contains 14 mechanisms: simple priority, priority with exception and 12 majority mechanisms that differ in how ties are broken.

We demonstrate the efficacy of our model by analyzing several existing mechanisms (§VI), including the popular cryptocurrency wallet Argent [4] used by over a million users [16]. Argent lets the user select m key guardians (e.g., friends) and keeps one key on the user's phone for a total of $m+1$ credentials. It provides functionality to transfer funds, add / delete any credential and lock the wallet in case of suspected failure. We find that if $m = 1$, the profile only has three scenarios, i.e., worse than our maximal priority mechanism. A similar result also holds for other m. Replacing the logic to use one of our maximal mechanisms would not only improve the security of Argent but also lead to a significantly simpler design. Our Solidity prototype of the priority mechanism (§C) is practical, requiring 210k gas for the most costly function (excluding deployment), compared to 150k for Argent ($7.6 and $5.4, resp., assuming a gas price of 30gwei and an Ethereum price of $1200).

Analyzing the 2-credential mechanism of Paralysis Proofs [13] we see it does better with a profile of five scenarios, but still lower than our maximal mechanism.

Our analysis emphasizes the criticality of ensuring the reliability and synchrony of the mechanism-user channel (§VII). In the context of cryptocurrencies, users can use multiple devices to monitor the blockchain. For services that rely on email or text notifications to the user, we propose to make such notifications "sticky", so an attacker that gains temporary access to user devices cannot delete them and break the communication channel.

In summary, our main contributions are:

1) A definition of the authentication problem, in particular under synchrony,
2) a probability-agnostic metric of the security level of an authentication mechanism, namely security profiles,
3) tight upper bound on the profile size of any mechanism,
4) novel n-credential mechanisms that are maximally secure,
5) the complete sets of maximal mechanisms for $n \leq 3$, and
6) security analysis of deployed and theoretical mechanisms with concrete improvement proposals.

II. Related work

To the best of our knowledge, previous work did not provide a general definition of the authentication problem.

The closest work is Eyal's cryptocurrency wallet design [11] (from which we borrow the definition of a scenario). He investigates what the best mechanism is, but in a restrictive setting with only boolean formulae, i.e., one-shot mechanisms. The maximal mechanisms we introduce achieve better security due to the use of interactivity.

Hammann et al. [12] use a similarly restrictive model, although their focus is orthogonal to ours, namely, to uncover vulnerabilities arising from the links between different mechanisms a user uses.

There is substantial interest in analyzing the security of multi-factor authentication mechanisms. For example, Jacomme et al. [17] and Barbosa et al. [18] analyze Google's 2FA and Fido's U2F. But they do not propose generic frameworks; moreover, these mechanisms are also one-shot.

Elaborate authentication mechanisms are common for cryptocurrency assets [19], [8], [10], [20], even interactive schemes like CoinVault [21] that inspired this work and SmartCustody [22]. This demonstrates the efficacy of interactive schemes in this context, but also the need for analysis, as we demonstrate by analyzing and proposing improvements to the popular Argent [4] in §VI-A.

Paralysis Proofs [13] leverages interactivity to deal with credential loss; similar ideas were recently proposed in the Bitcoin community [22]. We discuss this approach in §VI-C and show that our mechanisms achieves better security.

Vaults [15] (the first interactive cryptocurrency wallet design to our knowledge) and KELP [14] leverage interactivity to deal with key loss and leakage, respectively. However, they operate in more nuanced models that ours: KELP [14] considers partial knowledge (user knows about a lost key before anyone else), whereas we only consider complete knowledge. Vaults [15] treats the case of neither party being able to withdraw the funds differently than us (we assume the attacker wins). Both works consider specific mechanisms whereas we find bounds and maximal mechanisms.

III. Model

We define the authentication problem in this section. We describe the participants and communication (§III-A), credentials (§III-B), the authentication mechanism automaton (§III-C), and executions (§III-D). Finally we explain mechanism security profiles with which we evaluate mechanisms (§III-E).

A. Participants, time and network model

The system comprises a *user* U, an *attacker* A (also referred to as the *adversary*) and an *authentication mechanism* M.

An execution begins with nature assigning distinct *player identifiers* to the user and to the attacker from the set $I = \{\mathsf{id}_0, \mathsf{id}_1\}$. The player identifiers can be viewed as temporary identifiers akin to a cookie used to identify a website visitor during a single session.

During the execution, both parties interact with the mechanism by sending and receiving messages.

Time progresses in steps. The network is *synchronous*: if a message is sent in time step t, it reaches the recipient in step $t+1$. All the communication channels are *confidential*.

The mechanism can decide either id_0 or id_1. This is a one-time irrevocable operation corresponding to, say, allowing someone to withdraw money out of a bank account.

The adversary controls the ordering of *all messages within a time step*. That is, if both user and attacker send messages to the mechanism at the same time step, the attacker can choose which of the two is received first.

B. Messages, credentials and scenarios

During the execution, both parties interact with the mechanism by sending messages. Each message carries one or more credentials. Formally, a message m sent by U or A consists of a player identifier p and a set of credentials C, i.e., $m = (p, C)$. The identifier p must be an element of the set of all identifiers, i.e., $p \in I$. The set of credentials C is a subset of the set of all n credentials recognized by the mechanism M, i.e., $\mathcal{C}_{\mathsf{all}} = \{c_1, c_2, \ldots, c_n\}$ and $C \subseteq \mathcal{C}_{\mathsf{all}}$.

Each credential c_i is in one of four *states* [11]: (1) *Safe:* Only the user has it, (2) *Lost:* No one has it, (3) *Leaked:* Both the user and the adversary have it, or (4) *Stolen:* Only the adversary has it.

Denote the state of credential c_i by σ_i, so for all $1 \leq i \leq n : \sigma_i \in \{\mathsf{safe}, \mathsf{loss}, \mathsf{leak}, \mathsf{theft}\}$. A *scenario* σ is a vector of the n credential states and $\Sigma = \{\mathsf{safe}, \mathsf{loss}, \mathsf{leak}, \mathsf{theft}\}^n$ is the set of all scenarios, i.e. $\sigma \in \Sigma$.

The *credential set* of the user (resp., attacker) contains all the credentials available to that party, and is defined as $C^U_\sigma = \{c_i | \sigma_i \in \{\mathsf{safe}, \mathsf{leak}\}\}$ (resp., $C^A_\sigma = \{c_i | \sigma_i \in \{\mathsf{leak}, \mathsf{theft}\}\}$).

At the start of an execution, nature picks a scenario $\sigma \in \Sigma$ and initializes U and A with the credential sets C^U_σ and C^A_σ, respectively. The scenario σ is *common knowledge*, i.e., the state of all n credentials is known to both parties.

A message m sent by user or attacker can only carry a subset of the credentials available to the sender, i.e., a user (resp., attacker) message can carry a set of credentials C s.t. $C \subseteq C^U_\sigma$ (resp., $C \subseteq C^A_\sigma$). All messages are processed by the mechanism M potentially leading to a change to its internal state.

Note that the fact that we do not allow an attacker to include a credential not assigned to it *does not* imply that we preclude the attacker from forging credentials. Credential forgery is modeled as leakage or theft.

Next we present the automaton followed by details of an execution and the winner definitions.

C. Authentication mechanism

The authentication mechanism is a deterministic finite one-clock automaton M, defined by the following elements:

Automaton states: $\mathcal{S}$ is a finite set of the *states* of the automaton. It includes a special *starting state* S and two disjoint sets of *final states* $\mathcal{S}^{\mathsf{fin}}_0$ and $\mathcal{S}^{\mathsf{fin}}_1$. The set $\mathcal{S}^{\mathsf{fin}}_0$ is the set of final states where the authenticator chooses id_0 and the set $\mathcal{S}^{\mathsf{fin}}_1$ is where it chooses id_1. The set $\mathcal{S}^{\mathsf{int}}$ denotes the set of non-final states, $\mathcal{S}^{\mathsf{int}} = \mathcal{S} \setminus (\mathcal{S}^{\mathsf{fin}}_0 \cup \mathcal{S}^{\mathsf{fin}}_1)$.

Automaton clock: A clock *clock* has some value $v \in \mathbb{Z}_{\geq 0}$. The initial state is $v = 0$, and v is incremented by 1 in each step. Furthermore, the automaton can reset the clock back to its initial state ($v = 0$).

A pair of an automaton state s and a clock state v is called an *extended state*. Note that we refer just to the automaton state s when using the word state.

Automaton transition: An automaton transition is a 6-tuple consisting of a source state, a destination state, three types of guards and a clock reset. A guard is a constraint specifying that a certain condition must be met for this transition to take place. The three types of guards and clock reset are as follows:

Player guard: $\mathcal{G}_{\mathsf{id}} = I \cup \{\perp\}$ is the set of *player identifier guards*. The presence of a player ID guard in a transition

indicates that only the specified player ID can execute that transition. A $\perp$ stands for "no player guard".

We use the notation $p \vdash g^{plr}$ to denote that a player $p \in I$ satisfies the guard g^{plr}. For example, $\mathsf{id}_0 \vdash g^{plr}$ only if $g^{plr} \in \{\mathsf{id}_0, \perp\}$. The symbol $\nvdash$ means the guard is not satisfied.

Credential guard: $\mathcal{G}_c$ is the set of *credential guards* that includes all non-constant monotone boolean formulae of the n credentials in the set $\mathcal{C}_{\mathsf{all}}$ and a special value $\perp$. For example, if $n = 2$, then $\mathcal{G}_c = \{c_1, c_2, c_1 \wedge c_2, c_1 \vee c_2, \perp\}$.

We say that a message $m \subseteq \mathcal{C}_{\mathsf{all}}$ satisfies a credential guard g^{cd} using the notation $m \vdash g^{cd}$ if the credentials in m satisfy the boolean formula g^{cd}. For example, if $g^{cd} = c_1 \vee c_2$ and $m = \{c_1\}$, then m satisfies g^{cd}, so $m \vdash g^{cd}$.

Clock guard: $\mathcal{G}_v$ is the set of *clock guards*. Each clock guard g^{clk} is a condition on the clock state v expressed through a comparison operator $\sim \in \{<, \leq, >, \geq, =\}$ and a natural number $l \in \mathbb{N}$. That is, a clock guard is of the form $(v \sim l)$. A transition is allowed to have "no clock guards" represented by $\perp$.

For example, a guard $g^{clk} = (v < 5)$ specifies that the clock state v must be less than 5 when the transition is being taken. Like before, $v \vdash g^{clk}$ denotes that the clock state v satisfies the guard g^{clk}.

Clock reset: $\mathcal{R} = \{\mathsf{True}, \mathsf{False}\}$ represents the reset of the clock state or no reset. If a transition with a reset is taken, the clock is reset back to its initial state, i.e., we set $v \leftarrow 0$ at the end of the step.

Automaton definition: In summary, $\mathcal{D} \subseteq \mathcal{S}^{\mathsf{int}} \times \mathcal{G}(I) \times \mathcal{G}_c \times \mathcal{G}(v) \times \mathcal{R} \times \mathcal{S}$ is the set of transitions of M. And an automaton M is a 6-tuple given by $M = (\mathcal{C}_{\mathsf{all}}, \mathcal{S}, clock, \mathcal{D}, \mathcal{S}_0^{\mathsf{fin}}, \mathcal{S}_1^{\mathsf{fin}})$.

Recall that we consider a deterministic automaton. Determinism means that for all states $s \in \mathcal{S}$, all clock states $v \in \mathbb{Z}_{\geq 0}$, all sets of credentials $C \subseteq \mathcal{C}_{\mathsf{all}}$, and all player identifiers $p \in I$, there exists at most one transition $(s, g^{plr}, g^{cd}, g^{clk}, r, s') \in \mathcal{D}$ such that all the guards are satisfied, i.e., $p \vdash g^{plr}$, $C \vdash g^{cd}$ and $v \vdash g^{clk}$.

Remark: We use the notation $\mathcal{M}_n$ to denoted the set of *all* n-credential mechanisms.

D. Execution, player strategies and scenarios

The details of how an execution unfolds are specified in Fig. 1 and are explained now.

An execution begins by the nature assigning user identifiers arbitrarily. It picks $\gamma \in \{0, 1\}$ and assigns the identifier id_γ to the user and $\mathsf{id}_{1-\gamma}$ to the attacker. We refer to γ as nature's strategy. Then both parties interact with the automaton by sending messages carrying credentials.

Let the extended state of the automaton at time t be (s_t, v_t). Initially, $t = 0$, and the automaton is at it's start state S, i.e., $s_0 \leftarrow S$, and the clock is at zero, i.e., $v_0 \leftarrow 0$.

The automaton's state changes only if it receives a message m that satisfies the player, clock and credential guards of an outgoing transition from its current state s_t. Whenever a state change happens, the mechanism sends a message to both parties updating them about it.

Nature: Nature chooses a scenario $\sigma \in \Sigma$ and initializes U (resp., A) with the set of credentials C_σ^U (resp., C_σ^A). Nature picks $\gamma \in \{0, 1\}$ arbitrarily and assigns player identifiers using γ: $\mathsf{id}_U \leftarrow \mathsf{id}_\gamma$, $\mathsf{id}_A \leftarrow \mathsf{id}_{1-\gamma}$.

Mechanism M**:** The extended state of the automaton at time t is given by (s_t, v_t). Initially, $t \leftarrow 0$, $s_0 \leftarrow S$ and $v_0 \leftarrow 0$.

for t **in** $\{1, 2, \dots\}$:

$s_t \leftarrow s_{t-1}$, $v_t \leftarrow v_{t-1} + 1$.

Let the set of messages received at time t be $M = \{(p_i, C_i)\}_{i=1}^k$.

for (p, C) **in** M:

If $\exists (s, g^{plr}, g^{cd}, g^{clk}, r, s') \in \mathcal{D}$, $s = s_t$, $p \vdash g^{plr}$, $C \vdash g^{cd}$, and $v_t \vdash g^{clk}$, do the following:

1) update state: set $s_t \leftarrow s'$, and if $r = \mathsf{True}$, reset the clock $v_t \leftarrow 0$.
2) send the message {"new state", (s_t, v_t)} to id_0 and id_1.
3) if $s_t \in \mathcal{S}_0^{\mathsf{fin}}$ (resp., $s_t \in \mathcal{S}_1^{\mathsf{fin}}$), *decide* the execution winner is id_0 (resp., id_1).

Fig. 1: System execution

The mechanism processes all messages received in each time step. Note that it is possible that more than one state change happens within a single time step.

The execution ends as soon as a final state is reached, i.e., if $s_t \in \mathcal{S}_i^{\mathsf{fin}}$, then the winner is the player with identifier id_i.

Trace and run: We define three terms: *trace*, *run* and *extended run*, useful to talk about a finished execution. A trace $w = \{s[0], s[1], \dots, s[o]\}$ tracks all the automaton state changes where $s[i]$ is the state after the ith state change (as opposed to s_i which tracks the state after the ith time step); $s[0] = S$ and $s[o] \in (\mathcal{S}_0^{\mathsf{fin}} \cup \mathcal{S}_1^{\mathsf{fin}})$ is a final state. And a run $r = \{(p_1, m_1, t_1), (p_2, m_2, t_2), \dots, (p_o, m_o, t_o)\}$ tracks the corresponding sequence of (timed) player-messages that led to the above state changes, where all $p_i \in I$, all $m_i \in \mathcal{C}_{\mathsf{all}}$ and $(t_1, t_2, \dots)$ is a sequence of non-decreasing positive timestamps. Finally, an extended run tracks $r_{\mathsf{ext}} = \{(p_1, m_1, t_1), (p_2, m_2, t_2), \dots, (p_q, m_q, t_q)\}$ tracks *all* the messages received, not just the ones that lead to a state change.

User strategy: The user *strategy* S^U is a specification of how the user responds in any given (extended) state. We formalize it below.

Definition 1 (Messaging strategy)**.** *A messaging strategy for the user (resp., attacker) is a function of the extended state $e = (s, v)$, $S(e)$ that returns $\perp$ or an ordered list of credential sets $\{C_k\}$ s.t. $\forall k, C_k \subseteq C_\sigma^U$ (resp., $C_k \subseteq C_\sigma^A$).*

The output of a messaging strategy is a list of credential sets. Equivalently, it can be viewed as a list of messages with each message carrying a credential set. Note that we allow a player to send multiple messages in a single time step.

We say that a messaging strategy S is *playable* for the user or attacker under a scenario σ if it satisfies Definition 1.

Attacker strategy: The attacker's strategy S^A consists of two components: a messaging strategy S_{msg}^A and an *ordering strategy* S_{ord}^A, i.e., $S^A = (S_{\mathsf{msg}}^A, S_{\mathsf{ord}}^A)$. The latter captures how an attacker orders messages sent in the same time step.

Two examples of reordering strategies are an *attacker-first* strategy Ord^A, where the attacker orders it's own messages before the user's, and a *user-first* strategy Ord^U, where the attacker orders user's messages first. Unless specified otherwise, attacker employs Ord^A.

Execution: An execution E is fully defined by an automaton M, a scenario σ, player strategies S^U and S^A, and the nature's strategy γ, i.e., $E = (M, \sigma, S^U, S^A, \gamma)$.

Definition 2 (Execution winner)**.** *Given an execution $E = (M, \sigma, S^U, S^A, \gamma)$, the winner of an execution is the player decided by the automaton M, or the adversary A if M never decides. We denote $Win^{exec}(E) \in \{U, A\}$ accordingly.*

An execution E is said to *finish* if the automaton decides. As defined above, if E does not finish, then the attacker wins.

Note that the winner of two executions played with the same player strategies can differ due to changes in nature's strategy.

Success: A mechanism M is said to *succeed* or *fail* in a given scenario σ, denoted by $Suc(M, \sigma) \in \{\mathsf{True}, \mathsf{False}\}$ respectively. Success implies that the attacker cannot win in even a single execution, or conversely, that the user wins in all executions. That is, the user knows a *winning strategy* S^U_{win} that allows it to win in all executions irrespective of the nature's and attacker's strategies.

Definition 3 (Success)**.** *Given a mechanism M and a scenario σ, we say that the mechanism is* successful *if the user consistently wins against any attacker's strategy and nature's strategy, i.e., $Suc(M, \sigma) = \mathsf{True}$ if $\exists S^U_{\mathsf{win}}, \forall S^A, \forall \gamma$, $E = \{M, \sigma, S^U_{\mathsf{win}}, S^A, \gamma\}$ and $Win^{exec}(E) = U$. Otherwise, the mechanism fails and $Suc(M, \sigma) = \mathsf{False}$.*

We refer to the tuple of a mechanism and a scenario as an extended scenario denoted by $X = (M, \sigma)$. Denote $Win_X(S^U, S^A) = \text{U}$ if the strategy S^U wins for the user in *all executions* where the attacker employs S^A. On the other hand, $Win_X(S^U, S^A) = \text{A}$ means that S^A wins for the attacker in *at least one execution* where the user employs S^U.

A strategy S^U_{win} is a winning strategy for the user if it wins against any attacker strategy, i.e., $\forall S^A : Win_X(S^U_{\mathsf{win}}, S^A) = \text{U}$. Similarly, S^A_{win} is a winning strategy for the attacker if $\forall S^U : Win_X(S^U, S^A_{\mathsf{win}}) = \text{A}$.

We conclude this section with a few remarks.

Parallel executions: Note that multiple executions are allowed to run in parallel at the same time although we do not explicitly model it.

State updates: Recall that we assume that any state change is immediately informed to both the user and attacker. This is both a strict and conservative assumption: strict (resp., conservative) because the user (resp., attacker) knows all the state updates, including any authentication attempts by the attacker (resp., user).

E. Mechanism profiles

A mechanism's profile is a concise representation of the security level it achieves. It contains all the scenarios in which the mechanism succeeds.

Notation	Description
U, A	User, Attacker
$\mathcal{C}_{\mathsf{all}} = \{c_1, \ldots, c_n\}$	The n credentials
σ, $\overline{\sigma}$	State vector and its complement
$\sigma_i \in \{\mathsf{safe}, \mathsf{loss}, \mathsf{leak}, \mathsf{theft}\}$	State of c_i
σ^U_i, σ^A_i	Does the user, attacker have c_i?
C^U_σ / C^A_σ	Credentials of user / attacker in σ
$\Sigma_{\mathsf{ws}}, \Sigma_{\mathsf{ns}}$	All with-safe, no-safe scenarios
$\Sigma_{\mathsf{wt}}, \Sigma_{\mathsf{nt}}$	All with-theft, no-theft scenarios
$I = \{\mathsf{id}_0, \mathsf{id}_1\}$	Player identifiers
$M = (\mathcal{C}_{\mathsf{all}}, \mathcal{S}, clock, \mathcal{D}, \mathcal{S}^{\mathsf{fin}}_0, \mathcal{S}^{\mathsf{fin}}_1)$	Automaton / Mechanism
$\mathcal{S}^{\mathsf{fin}}_0 / \mathcal{S}^{\mathsf{fin}}_1$	Final states where id_0 / id_1 wins
$\mathcal{G}_c, \mathcal{G}_v, \mathcal{G}_{\mathsf{id}}$	Credential, clock and player guards
$C \vdash g^{cd}$	Credential guard
$clock, v$	The clock and its state
$g^{clk} := v \sim l$, $v \vdash g^{clk}$	Clock guard
$r \in \{\mathsf{True}, \mathsf{False}\}$	Clock reset
$(s, g^{plr}, g^{cd}, g^{clk}, r, s') \in \mathcal{D}$	A 6-tuple edge in the automaton
$Suc(M, \sigma)$	Is the mechanism successful in σ?
S^U, S^A	Strategies of user and attacker
$Win^{\mathsf{exec}}(E)$	Winner of an execution
$Win_\sigma(S^U, S^A)$	Winner under specified strategies
$w = \{s_i\}^n_{i=0}$	The trace of a finished execution
$r = \{p_i, m_i, t_i\}^n_{i=1}$	The run of a finished execution
$prof(M)$	Security profile of the scheme M
$P(n)$	Bound on the profile size
$\mathcal{M}_n$	All n-credential mechanisms
$\mathcal{O}_n$	Complete maximal set of n-cred mechs

TABLE I: Notation

Definition 4 (Profile)**.** *Given a mechanism M, it's profile $prof(M)$ is the set of all scenarios where M succeeds, i.e., $prof(M)$ contains any $\sigma \in \Sigma$ such that $Suc(M, \sigma) = \mathsf{True}$.*

The profile can also be viewed as an n-dimensional matrix where n is the number of credentials in M. Each dimension has four rows corresponding to the four states that a credential can be in. Each matrix cell represents a distinct scenario and the value in the cell is 1 if the scenario is in the profile or 0 otherwise. An example matrix is in Table II.

The set of all n-credential mechanisms is a partially ordered set. Given two mechanisms M_1 and M_2 using the same number of credentials, we can compare them by looking at their corresponding profiles to arrive at one of the relations: *equivalent*, *better*, *worse* or *incomparable*, defined below.

Definition 5 (Equivalent mechanisms)**.** *Two n-credential mechanisms $M, M' \in \mathcal{M}_n$ are* equivalent*, denoted by $M \cong M'$, if one's profile can be obtained from the other's by simply permuting the credential set. Formally, given a permutation $\pi : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ and a scenario σ, define the permuted scenario σ^π by $\sigma^\pi_i = \sigma_{\pi(i)}$. Then, M and M' are equivalent if $\exists \pi : \bigcup_{\sigma \in prof(M)} \sigma^\pi = prof(M')$.*

Remark: Two mechanisms with same profiles are equivalent due to the trivial permutation $\pi(x) = x$.

Given a mechanism M, the set of all mechanisms obtained by permuting the credential set of M is called the *isomorphic expansion* of M denoted by $expand(M) = \{M, M', \ldots\}$.

Definition 6 (Better / Worse mechanisms)**.** *A mechanism M_1 is* better *than another mechanism M_2, denoted by $M_1 \succ M_2$, or M_2 is* worse *than M_1, denoted by $M_2 \prec M_1$, if $\exists M_3 \cong M_2$ such that prof$(M_1) \supset$ prof(M_3).*

Remark: We denote *better than or equivalent to* by $M_1 \succeq M_2$, which means that either $M_1 \succ M_2$ or $M_1 \cong M_2$. ($\preceq$ is defined analogously.)

Definition 7 (Incomparable mechanisms)**.** *A mechanism M_1 is* incomparable *to another mechanism M_2, denoted by $M_1 \nsim M_2$, if $M_1 \not\succeq M_2 \wedge M_1 \not\prec M_2$.*

We now define *maximally secure* mechanisms, which are mechanisms such that no other mechanism is better than them.

Definition 8 (Maximal mechanism)**.** *An n-credential mechanism $M \in \mathcal{M}_n$ is said to be maximally secure if for all n-credential mechanisms $M' \in \mathcal{M}_n$: either $M' \preceq M$ or $M' \nsim M$.*

Note that the above definition leaves open the possibility that multiple maximal mechanisms exist. It can be observed that any two maximal mechanisms, if they exist, must be either equivalent or incomparable to each other.

IV. Bounded-delay maximal mechanisms

We present an upper bound on the profile set size, i.e., given n, we find the maximum number of scenarios in which an n-credential mechanism succeeds (§IV-A). Next, we introduce bounded-delay mechanisms, which decide within a bounded time. We prove that any bounded-delay mechanism satisfying three general properties is maximal (§IV-B). Finally, we present some algorithms that produce maximal n-credential mechanisms (§IV-C).

A. Maximality bounds

To establish a bound on the profile size, we first prove a few important results.

The first result is about *complement scenarios*. The complement of a scenario σ switches all the safe credentials to theft and vice versa, while leaked and lost credentials remain the same. More formally:

Definition 9 (Complement scenario)**.** *Given a scenario σ, we define its* complement scenario *by its elements:*

$$\overline{\sigma}_i = \begin{cases} \textit{theft} & \sigma_i = \textit{safe} \\ \textit{safe} & \sigma_i = \textit{theft} \\ \sigma_i & \textit{otherwise.} \end{cases} \tag{1}$$

First, any two executions with the same extended run will have the same winning player identifier.

Lemma 1. *If two executions E_1 and E_2 using the same mechanism M have the same finite extended run, then the winner of both executions has the same player identifier.*

Proof. Note that the lemma refers to executions that have a *finite* extended run, i.e., we are only considering executions that finish.

If two executions E_1 and E_2 share the same extended run and use the same automaton, then they will have the same trace because we are using a deterministic automaton. In particular, given an (extended) state of the automaton, processing an input message m from a player p always causes the same state change. Therefore, E_1 and E_2 will have the same traces, and consequently the same final state. Denote the final state by s. And depending on whether $s \in \mathcal{S}_0^{\mathsf{fin}}/\mathcal{S}_1^{\mathsf{fin}}$, either the player $\mathsf{id}_0/\mathsf{id}_1$ wins in both the executions. □

Next we prove that if the user wins in all executions where the attacker employs a specific strategy, then switching the messaging strategies allows the attacker to win an execution.

Lemma 2. *Let there be a mechanism M, two extended scenarios $X_1 = (M, \sigma_1)$, $X_2 = (M, \sigma_2)$, and two messaging strategies S_1, S_2 such that S_1 is playable by U in σ_1 and by A in σ_2 and S_2 is playable by U in σ_2 and by A in σ_1. There exists a reordering strategy $\overline{S^A_{\mathsf{ord}}}$ such that the two attacker strategies $S_1^A = (S_2, Ord^A)$, $S_2^A = (S_1, \overline{S^A_{\mathsf{ord}}})$ satisfy $Win_{X_1}(S_1, S_1^A) = U \implies Win_{X_2}(S_2, S_2^A) = A$.*

Proof. $Win_{X_1}(S_1, S_1^A) = \mathrm{U}$ implies that the user wins in all executions where the attacker's strategy is S_1^A. Consider an execution $E = (M, \sigma_1, S_1, (S_2, Ord^A), 0)$, i.e., the user is assigned id_0. We have $Win^{\mathsf{exec}}(E) = \mathrm{U}$. Let the extended run of E be r_{ext}.

Consider another execution $E' = (M, \sigma_2, S_2, (S_1, \overline{S^A_{\mathsf{ord}}}), 1)$, i.e., the attacker is assigned id_0 and $\overline{S^A_{\mathsf{ord}}} \leftarrow Ord^U$ is the user-first ordering strategy. We first handle the case where the mechanism in execution E' does not decide, i.e., E' runs forever. By definition, $Win^{\mathsf{exec}}(E') = \mathrm{A}$, and we are done.

Now assume E' finishes; let its extended run be r'_{ext}. We now prove that the two extended runs are same, i.e., $r_{\mathsf{ext}} = r'_{\mathsf{ext}}$. First, note that in both executions, the player id_0 uses the messaging strategy S_1 and id_1 uses the messaging strategy S_2. Secondly, since the attacker's reordering strategy prioritizes attacker in E_1 and user in E_2, the messages sent by id_1 are prioritized in both executions. And therefore, the messages received by the authenticator in any given time step are the same in both executions including the order of messages received within a time step. Therefore $r_{\mathsf{ext}} = r'_{\mathsf{ext}}$.

Lemma 1 completes the proof. It says that the winner of two executions with the same extended run has the same player identifier. And since player id_0 (user) wins E_1, attacker wins execution E_2. □

Finally, we prove that if a mechanism succeeds in a scenario σ, then it fails in the complement scenario $\overline{\sigma}$.

Theorem 1. *If a mechanism M is successful in a scenario σ, then it must be unsuccessful in the complement scenario $\overline{\sigma}$: $Suc(M, \sigma) = \mathit{True} \implies Suc(M, \overline{\sigma}) = \mathit{False}$.*

The intuition behind the proof is that the attacker will be able to employ in $\overline{\sigma}$ the same messaging strategy that the user uses in σ. Because the user's strategy wins in all executions against any attacker strategy in σ, we show that the attacker's strategy wins in at least one execution of $\overline{\sigma}$.

Proof. Let S^U denote a winning strategy of the user in the extended scenario $X = (M, \sigma)$. Since S^U wins in all executions, we have $\forall S^A : \mathit{Win}_X(S^U, S^A) = \mathrm{U}$.

We produce a winning strategy $\bar{S}^A$ for the attacker in $\bar{X} = (M, \overline{\sigma})$ based on S^U. To do this, we need to show two things, namely, that S^U is a playable messaging strategy for the attacker in the complement scenario $\overline{\sigma}$, and that it succeeds in at least one execution.

The first follows in a straightforward way from the definition of a complement. It is easy to see that $C^U_\sigma = C^A_{\overline{\sigma}}$. Therefore, any user strategy in the scenario σ is a playable messaging strategy by the attacker in the complement scenario $\overline{\sigma}$. In particular, S^U is a valid messaging strategy for the attacker in $\overline{\sigma}$.

Next we need to show that the attacker can succeed with the messaging strategy S^U in the complement scenario. Recall that Ord^A denotes the attacker-first reordering strategy. Let the attacker's strategy be

$$\bar{S}^A \leftarrow \{S^U, \mathit{Ord}^A\}.$$

We need to show that no user strategy succeeds in all executions of $\bar{X} = (M, \overline{\sigma})$, i.e., $\nexists \bar{S}^U : \mathit{Win}_{\bar{X}}(\bar{S}^U, \bar{S}^A) = \mathrm{U}$.

We prove by contradiction. Assume the existence of a winning user strategy $\bar{S}^U$, i.e.,

$$\mathit{Win}_{\bar{X}}(\bar{S}^U, \bar{S}^A) = \mathrm{U}.$$

We will use the successful strategy $\bar{S}^U$ to devise a strategy S^A for the attacker in the original scenario.

Lemma 2 completes the proof. It shows that if a user strategy S_1 wins in all executions against an attacker strategy S_2 in a scenario σ_1, then interchanging players' messaging strategies allows the attacker to win an execution in a different scenario σ_2 assuming all strategies are playable for the respective players in the respective scenarios.

Our idea is to set $\sigma_1 = \overline{\sigma}$, $\sigma_2 = \sigma$, $S_1 = \bar{S}^U$ and $S_2 = S^U$ in Lemma 2. Before we apply this lemma, we first need to show that S_1 ($\bar{S}^U$) is a playable messaging strategy for the attacker in σ_2 (σ). But this follows because $C^U_{\overline{\sigma}} = C^A_\sigma$, which can be seen easily.

Lemma 2 guarantees the existence of a ordering strategy S^A_{ord} such that

$$\mathit{Win}_{\bar{X}}(\bar{S}^U, \bar{S}^A) = \mathrm{U} \implies \mathit{Win}_X(S^U, S^A) = \mathrm{A}.$$

This completes the proof because we found an attacker strategy S^A that wins an execution against the user winning strategy S^U in the original extended scenario X. □

We now prove that if no credential is safe in a scenario σ, then no mechanism succeeds in σ.

Theorem 2. *Given n credentials and a scenario σ^{bad} where no key is safe, i.e., $\sigma^{\mathsf{bad}} \in \{\mathsf{loss}, \mathsf{leak}, \mathsf{theft}\}^n$, then $\nexists M : \mathit{Suc}(M, \sigma^{\mathsf{bad}}) = \mathsf{True}$.*

Proof. By contradiction, suppose there exists a mechanism M where the mechanism succeeds in a bad scenario, i.e., $\mathit{Suc}(M, \sigma^{\mathsf{bad}}) = \mathsf{True}$. Let the extended scenario be $X = (M, \sigma^{\mathsf{bad}})$. Let the winning strategy of user be S^U, so for all S^A, $\mathit{Win}_X(S^U, S^A) = \mathrm{U}$.

Let $S^*_1 \leftarrow \{S^U, \mathit{Ord}^A\}$. We have $\mathit{Win}_X(S^U, S^*_1) = \mathrm{U}$ since S^U is a winning strategy for the user by assumption.

We now apply Lemma 2. Let $\sigma_1 = \sigma$, $\sigma_2 = \sigma$, $S_1 = S^U$, and $S_2 = S^U$. A pre-condition for Lemma 2 is that S^U be playable by both user and attacker in σ. This is true because we are operating in the restricted class of bad scenarios where no credential is safe, so $C^U_\sigma \subseteq C^A_\sigma$. Consequently, for any set of credentials C, if $C \subseteq C^U_\sigma$ then $C \subseteq C^A_\sigma$. So any strategy employed by the user is also a playable strategy for the attacker, including S^U.

Applying Lemma 2 gives $\mathit{Win}_X(S^U, S^*_1) = \mathrm{U} \implies \mathit{Win}_X(S^U, S^*_2) = \mathrm{A}$ where $S^*_2 = \{S^U, S^A_{\mathsf{ord}}\}$ for some S^A_{ord}.

Since $\mathit{Win}_X(S^U, S^*_1) = \mathrm{U}$, it must be that $\mathit{Win}_X(S^U, S^*_2) = \mathrm{A}$. That is, S^*_2 is a winning strategy for the attacker, which contradicts the assumption that S^U is a winning strategy for the user. Hence proved. □

We now prove a bound on how secure an authentication mechanism can be. Before doing so, we categorize scenarios.

Definition 10 (With-safe / With-theft scenarios)**.** *Define a* with-safe scenario *(resp.,* with-theft scenario*) as a scenario in which at least one credential is safe (resp., theft). Denote the set of all with-safe (resp., with-theft) scenarios by Σ_{ws} (resp., Σ_{wt}).*

Definition 11 (No-safe / No-theft scenarios)**.** *Define a* no-safe scenario *(resp.,* no-theft scenario*) as a scenario in which no credential is safe (resp., theft). Denote the set of all no-safe (resp., no-theft) scenarios by Σ_{ns} (resp., Σ_{nt}).*

Note that $\overline{\Sigma_{\mathsf{ws}}} = \Sigma_{\mathsf{ns}}$ and $\overline{\Sigma_{\mathsf{wt}}} = \Sigma_{\mathsf{nt}}$ where $\overline{T}$ denotes the complement set of T.

Given n credentials, we define two bounds that any authentication mechanism M must adhere to:

1) $P(n)$ is an upper bound on the number of scenarios M succeeds in, i.e., the profile set size.
2) $Q(n) = Q_{\mathsf{os}}(n) + Q_{\mathsf{ns}}(n)$ is a lower bound on the number of scenarios M fails where:
   a) $Q_{\mathsf{os}}(n)$ is a lower bound on the number of with-safe (os) scenarios M fails.
   b) $Q_{\mathsf{ns}}(n) = 3^n$ is the total number of no-safe (ns) scenarios since M fails in them by Theorem 2.

Theorem 3. *The upper bound on the size of the profile is $P(n) = (4^n - 2^n)/2$.*

The proof idea is as follows. We first compute the minimal number of with-safe scenarios in which the attacker succeeds, i.e., $Q_{\mathsf{os}}(n)$. To arrive at this lower bound, we limit our search to the set of scenarios that are both with-safe and with-theft ($\Sigma_{\mathsf{ws}} \cap \Sigma_{\mathsf{wt}}$). We show that the attacker wins in at least half of the with-safe-with-theft scenarios. This gives us a lower bound on the number of with-safe scenarios where the attacker succeeds ($Q_{\mathsf{os}}(n)$), and consequently an upper bound on the profile size ($P(n)$).

Proof. We treat the case of $n = 1$ separately first. In this case, there are only 4 scenarios in total (safe, leak, theft, loss) and $Q_{\mathsf{ns}}(1) = 3$, so the adversary succeeds in 3 of the 4 scenarios. The bound on profile size $P(1) = 1$ follows.

The rest of the proof corresponds to the case of $n > 1$. We count the number of with-safe-with-theft scenarios. A with-safe-with-theft scenario σ satisfies $\exists i, j : \sigma_i = \mathsf{safe} \wedge \sigma_j = \mathsf{theft}$. Denote the set of all with-safe-with-theft scenarios by $\Sigma_{\mathsf{wswt}} = \Sigma_{\mathsf{ws}} \cap \Sigma_{\mathsf{wt}}$.

We find the size of its complement set $\overline{\Sigma_{\mathsf{wswt}}}$ and subtract it out from the total number of scenarios. The total number of scenarios is 4^n as each credential can be in one of four states (safe, leak, theft, loss).

By De-Morgan's law, we have $\overline{\Sigma_{\mathsf{wswt}}} = \Sigma_{\mathsf{ns}} \cup \Sigma_{\mathsf{nt}}$. To count the size of $\overline{\Sigma_{\mathsf{wswt}}}$, we use the formula $|\overline{\Sigma_{\mathsf{wswt}}}| = |\Sigma_{\mathsf{ns}}| + |\Sigma_{\mathsf{nt}}| - |\Sigma_{\mathsf{ns}} \cap \Sigma_{\mathsf{nt}}|$. We have $|\Sigma_{\mathsf{ns}}| = |\Sigma_{\mathsf{nt}}| = 3^n$ and $|\Sigma_{\mathsf{ns}} \cap \Sigma_{\mathsf{nt}}| = 2^n$ (as each credential can be either leak or theft). So the number of with-safe-with-theft scenarios $|\Sigma_{\mathsf{wswt}}| = 4^n - 2 \cdot 3^n + 2^n$.

We now show that that the mechanism cannot succeed in more than half of the with-safe-with-theft scenarios. Applying Theorem 1 to with-safe-with-theft scenarios: for every with-safe-with-theft scenario $\sigma \in \Sigma_{\mathsf{wswt}}$ that the user wins in, the attacker wins in its complement $\overline{\sigma}$. And we have $\overline{\sigma} \in \Sigma_{\mathsf{wswt}}$ because a with-safe-with-theft scenario has (at least) one safe and one theft credential, and the safe, theft states are switched in the complement transform. Therefore, the user cannot win in more than half of the with-safe-with-theft scenarios, or conversely, the attacker succeeds in at least half of the with-safe-with-theft scenarios.

And the number of with-safe scenarios where A succeeds is at least

$$Q_{\mathsf{os}}(n) = |\Sigma_{\mathsf{wswt}}|/2 = \frac{4^n - 2 \cdot 3^n + 2^n}{2}. \tag{2}$$

The total number of scenarios where A succeeds is at least $Q(n) = Q_{\mathsf{os}}(n) + Q_{\mathsf{ns}}(n) = (4^n - 2 \cdot 3^n + 2^n)/2 + 3^n = (4^n + 2^n)/2$. And conversely, the maximum number of scenarios in which U succeeds is

$$P(n) = 4^n - Q(n) = 4^n - \frac{4^n + 2^n}{2} = \frac{4^n - 2^n}{2}. \tag{3}$$

□

Remark: Note that *not using* all the available credentials results in a non-maximal mechanism, as one might expect. Namely, if you take a maximal n-credential mechanism and ask whether it's also an maximal $(n + k)$-credential mechanism for some $k > 0$ where the k extra credentials are simply unused, the answer is no because $P(n + k) > 4^k P(n)$.

B. Generating maximal mechanisms

Our strategy for finding maximal mechanisms is to find mechanisms whose profile size is $P(n)$. This is because such a mechanism trivially satisfies the definition of security-maximality in Definition 8.

We now give a simple way of generating maximal n-credential mechanisms. These mechanisms wait for a bounded time, allowing both parties to submit credentials. We refer to the resulting mechanisms as *bounded-delay mechanisms*.

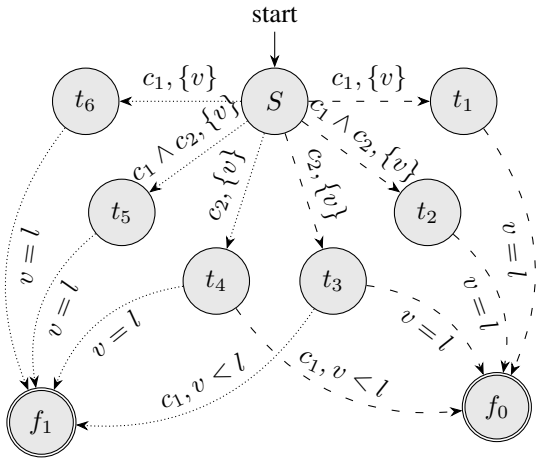


Fig. 2: A well-formed mechanism using 2 credentials c_1, c_2. Player id_0 (resp., id_1) can only take dashed (resp., dotted) edges and tries to reach f_0 (resp., f_1).

The idea behind these mechanisms is straightforward. The mechanism gives both parties an opportunity to submit a set of credentials. In particular, the mechanism starts a timer after receiving the first message, and the other party would need to submit its message before the timer ends. Finally, a deterministic *judging function* J selects the winner.

More formally, say the set of credentials submitted by the first player (id_0) is C_0 and the second player (id_1) is C_1 where $C_0, C_1 \subseteq \mathcal{C}_{\mathsf{all}}$. The judging function J takes the two sets of credentials as inputs and outputs the identifier of the successful player, i.e., $J(C_0, C_1) \mapsto \{\mathsf{id}_0, \mathsf{id}_1\}$.

1) Bounded-delay mechanisms: The bounded-delay mechanisms are realizable through the automaton model. Given a judging function J, the procedure below specifies how to construct a bounded-delay mechanism $M(J)$ through an example.

We explain the idea through an example. Given two credentials c_1, c_2, consider the judging function J that prioritizes c_1 over c_2. Figure 2 shows the automaton obtained after applying the below procedure. The high-level idea is as follows. The automaton has states in two levels below the start state. The intermediate level contains a state for each combination of player and set of credentials submitted, forming the children of the start state S. Then, for each intermediate state, we find all possible credential sets the second player can submit in order to win and add outgoing edges correspondingly. There are only two final states, one each for the respective players.

As shown in Fig. 2, the start state has six children corresponding to the 3 possible credential sets $(c_1, c_2, c_1 \wedge c_2)$ from the 2 parties (id_0, id_1). These six states form the intermediate states. Only two final states f_0 and f_1 exist. Player id_0 (resp., id_1) wins if the execution reaches f_0 (resp., f_1). Player guards

are depicted through dashed and dotted edges: id_0 can only take dashed edges while id_1 can only take dotted edges.

Given a judging function J defined over a set of credentials $\mathcal{C}_{\mathsf{all}}$, we describe an authentication mechanism $M = (\mathcal{C}_{\mathsf{all}}, \mathcal{S}, clock, \mathcal{D}, \mathcal{S}_0^{\mathsf{fin}}, \mathcal{S}_1^{\mathsf{fin}})$.

Let the set of all AND-credential-guards that use $\wedge$ connector only be G. $|G| = 2^n - 1$ as each credential can either be present or absent and we omit the ε-transition.

The set of all states $\mathcal{S}$ consists of $2 \cdot (2^n - 1)$ intermediate states and 2 final states. An intermediate state t is created for each player ID guard $g^{plr} \in I$ and credential guard $g_\gamma^{cd} \in G$, with a transition between the start state and this new state, $(S, g^{plr}, g_\gamma^{cd}, \bot, \mathsf{True}, t)$, i.e., no clock guard, with clock reset.

The two final states are f_0, f_1. The set $\mathcal{S}_0^{\mathsf{fin}}$ contains f_0 and the set $\mathcal{S}_1^{\mathsf{fin}}$ contains f_1.

The set of all transitions $\mathcal{D}$ consists of edges between the start and intermediate states (explained before) and those between the intermediate and final states (explained next).

There are two different types of transitions between the intermediate and final states. The first type allows the first-mover to win but only after some time elapses. For each intermediate state t, if the player identifier that submitted credentials before is $\mathsf{id}_\gamma \in I$, then there is a transition $(t, p, \bot, v = l, \mathsf{False}, f_\gamma)$, i.e., no credential guard.

The second type allows the other party to win, but only if they submit better credentials. Let C_γ denote the set of credentials submitted by the player id_γ (in g_γ^{cd}) to reach an intermediate state t. Let $p' = \mathsf{id}_{1-\gamma}$.

Find the set of credential guards G' that result in the second-mover winning, i.e., if $\gamma = 0$, find all $g_1^{cd} \in \mathcal{G}_c$ such that C_1 contains the credentials in g_1^{cd} and $J(C_0, C_1) = \mathsf{id}_1$. And if $p = \mathsf{id}_1$, find g_1^{cd} s.t. $J(C_1, C_0) = \mathsf{id}_0$. Add all such guards to G'. For each $g^{cd} \in G'$, there is a transition $(t, p', g^{cd}, v < l, \mathsf{False}, f_{1-\gamma})$.

2) Well-formed judging functions: We now prove that bounded-delay mechanisms derived from judging functions that satisfy three properties are maximal.

Definition 12 (ID-Agnostic (IA))**.** *A judging function J is ID-agnostic if the identifier assignment does not matter, i.e., if $C \neq C'$ then $(J(C, C') = \mathsf{id}_0) \Leftrightarrow (J(C', C) = \mathsf{id}_1)$.*

If a judging function J satisfies IA, then we can compare any two sets of credentials and say that one of them is better than the other. We use the notation $C \succ_J C'$ (subscript omitted when clear) to mean that C is *better* than C' according to J. Similarly, $C \succeq_J C'$ means that either the two credential sets are exactly equal, i.e., $C = C'$ or C is better than C'.

Note that we do not say anything about the case where $C = C'$ in the above definition, so a ID-agnostic function can choose to output either id_0 or id_1. All functions we present follow $\forall C : \mathsf{id}_0 \leftarrow J(C, C)$.

Definition 13 (Knowledge-rewarding (KR))**.** *A judging function is knowledge-rewarding if, when the credentials submitted by one party are a strict subset of those submitted by the other, then the latter succeeds, i.e., if $C' \subset C$ then $C' \prec C$.*

Definition 14 (Transitive knowledge-rewarding (TKR))**.** *A judging function is transitive knowledge-rewarding if additional credentials cannot weaken a player's strategy, i.e., if $C_0 \neq C_1 \neq C_2$, $C_0 \succ C_1$, and $C_1 \succ C_2$, then $C_0 \not\subseteq C_2$.*

Any judging function that satisfies these properties is said to be *well-formed* denoted by J, and the resulting mechanism $M(J)$ is a *well-formed mechanism*.

Definition 15 (Well-formedness)**.** *A judging function is said to be well-formed if it satisfies IA, KR and TKR.*

We now prove that any well-formed mechanism is maximal.

Theorem 4. *Any well-formed mechanism is maximally secure.*

We proceed in a few steps. Lemma 4 shows that the user can win in all with-safe-no-theft scenarios and Lemma 7 shows that the user can win in exactly half of the with-safe-with-theft scenarios. Using these two lemmas, Lemma 8 shows that the profile size of any n-credential well-formed mechanism is $P(n)$. This completes the proof because having a profile size of $P(n)$ implies maximality due to Theorem 3.

Before presenting the proofs, we define submit-early strategies and Lemma 3, which we use throughout the rest of the proofs. Note that all the lemmas in the rest of this section apply only to well-formed judging functions.

Definition 16 (Submit-early strategy)**.** *A submit-early strategy S_{early} is a strategy where the player sends a message m when the clock is at zero and does nothing thereafter, i.e.,*

$$S_{\mathsf{early}}(s, v) = \begin{cases} \{m\} & \text{if } s = S, v = 0 \\ \bot & \text{otherwise.} \end{cases}$$

Note: Denote a function $\Gamma(C)$ that returns the submit-early strategy where the message m contains the credentials C.

The next lemma says that if one party employs a submit-early strategy with credentials C, then the only way the other party wins is by submitting a set of credentials better than or equal to C.

Lemma 3. *Given a well-formed mechanism M and a scenario σ, if one party employs a submit-early strategy $S_{\mathsf{early}} = \Gamma(C)$, but the other party wins an execution, then the winning party must have submitted a message carrying better or equal credentials C', i.e., $C' \succeq C$.*

Proof. Say party 1 submits C as part of the submit-early strategy S_{early}, but party 2 wins an execution.

Irrespective of what party 2's strategy is, observe that it gets a chance to submit *at most one message* that causes a change in automaton's state. This is because of the way player guards are defined in the automaton M. Denote the credentials in this message by C'.

Two cases emerge based on the order in which the two sets of credentials, C and C', are processed by the automaton. But, irrespective of the order, party 2 wins only if $C' \succeq C$: IA guarantees that, if on the contrary $C' \prec C$, then the automaton would not prefer party 2 irrespective of the player identifier

(id_0 or id_1) assigned to it. So party 2 cannot win an execution, and therefore, it must be that $C' \succeq C$. □

Remark: Denote by $S^U_{\mathsf{all}} = \Gamma(C^U_\sigma)$ and $S^A_{\mathsf{all}} = \Gamma(C^A_\sigma)$ the submit-early strategy where all credentials owned by the party are submitted.

We now prove that well-formed mechanisms succeed in with-safe-no-theft scenarios.

Lemma 4. *Well-formed mechanisms succeed in with-safe-no-theft scenarios, i.e.,* $\forall \sigma \in \Sigma_{\mathsf{wsnt}}$, $\mathit{Suc}(M_{\mathsf{wf}}, \sigma) = \mathsf{True}$.

Proof. Assume for contradiction that the attacker wins in a with-safe-no-theft scenario $\sigma \in \Sigma_{\mathsf{wsnt}}$, i.e., $\sigma \in \Sigma_{\mathsf{ws}} \cap \Sigma_{\mathsf{nt}}$. This means that the attacker wins at least one execution against all user strategies.

But if the user follows the submit-early strategy S^U_{all}, then, due to Lemma 3, the attacker must know a set of credentials $C \subseteq C^A_\sigma$ such that $C \succeq C^U_\sigma$.

But since the scenario σ is a with-safe-no-theft scenario, we have $C^A_\sigma \subset C^U_\sigma$, and since $C \subseteq C^A_\sigma$, we conclude that $C \subset C^U_\sigma$.

But due to the knowledge-rewarding (KR) property, $C \subset C^U_\sigma$ implies $C \prec C^U_\sigma$, thereby contradicting the previous statement $C \succeq C^U_\sigma$. □

We now proceed to prove that the user can win in exactly half of the with-safe-with-theft scenarios Σ_{wswt}.

First, we present some useful lemmas. All the remaining lemmas in this section apply to with-safe-with-theft scenarios only.

We now prove that S^A_{all} is a winning strategy for the attacker provided that some other winning strategy exists.

Lemma 5. *Given a well-formed mechanism M, let the winning strategy of an attacker in a with-safe-with-theft scenario $\sigma \in \Sigma_{\mathsf{wswt}}$ be $S^A \neq S^A_{\mathsf{all}}$, then S^A_{all} is also a winning strategy. That is, if $X = (M, \sigma)$ and $\forall S^U : \mathit{Win}_X(S^U, S^A) = A$ then $\forall S^U : \mathit{Win}_X(S^U, S^A_{\mathsf{all}}) = A$.*

Proof. Assume for contradiction the existence of a user strategy S^U such that the strategy S^A_{all} never wins an execution. By Lemma 3, it must be that the user submits a credential set $C^U \subseteq C^U_\sigma$ that is better than or equal to C^A_σ, i.e., $C^U \succeq C^A_\sigma$. Furthermore, the inequality is strict, i.e., $C^U \succ C^A_\sigma$, because if $C^U = C^A_\sigma$ then since $C^U \subseteq C^U_\sigma$, we get $C^A_\sigma \subseteq C^U_\sigma$, which is not true for a with-safe-with-theft scenario $\sigma \in \Sigma_{\mathsf{wswt}}$.

To arrive at a contradiction, we start again from the assumption. Since S^A is a successful attacker strategy, it must win an execution against the submit-early strategy $S^* = \Gamma(C^U)$. And because of Lemma 3, the attacker must have submitted a credential set C^A such that $C^A \succeq C^U$.

We have two cases: $C^A = C^U$ or $C^A \succ C^U$. If $C^A = C^U$, since $C^U \succ C^A_\sigma$, we have $C^A \succ C^A_\sigma$. But KR says that if $C^A \subseteq C^A_\sigma$, then $C^A \preceq C^A_\sigma$, which leads to a contradiction.

Next, if $C^A \succ C^U$, since $C^U \succ C^A_\sigma$, the TKR property implies that $C^A \not\subseteq C^A_\sigma$. But we have a contradiction as $C^A \subseteq C^A_\sigma$ by definition. So we conclude that the strategy S^A_{all} is also winning. □

The next lemma proves that the strategy S^A_{all} is a winning strategy for the user in the complement scenario $\overline{\sigma}$.

Lemma 6. *If S^A_{all} is a winning strategy for the attacker in a with-safe-with-theft scenario $\sigma \in \Sigma_{\mathsf{wswt}}$, then the messaging component of S^A_{all} is also a winning strategy for the user in the complement with-safe-with-theft scenario $\overline{\sigma}$.*

Proof. Since S^A_{all} is a winning strategy in scenario σ, it must win against any user strategy, including S^U_{all}. By Lemma 3, since the user follows a submit-early strategy, the attacker must have submitted a set of credentials C such that $C \succeq C^U_\sigma$. Since the attacker also follows a submit-early strategy S^A_{all}, the attacker submits credentials exactly once. Therefore, it must be that $C = C^A_\sigma$ and hence $C^A_\sigma \succeq C^U_\sigma$. Again, the inequality is strict, i.e.,

$$C^A_\sigma \succ C^U_\sigma \tag{4}$$

since $C^A_\sigma \neq C^U_\sigma$ for a with-safe-with-theft scenario $\sigma \in \Sigma_{\mathsf{wswt}}$.

Assume for contradiction that $\bar{S}^U_{\mathsf{all}} = S^A_{\mathsf{all}}$ (we mean that $\bar{S}^U_{\mathsf{all}}$ is the messaging component of S^A_{all}) is not a winning strategy for the user in the complement scenario $\overline{\sigma}$. It means that there exists an attacker strategy $\bar{S}^A$ that wins against $\bar{S}^U_{\mathsf{all}}$ in some executions of $\overline{\sigma}$. Applying Lemma 3 again, the attacker must have submitted a set of credentials $C \subseteq C^A_{\overline{\sigma}}$ such that $C \succeq C^U_{\overline{\sigma}}$. Since $\overline{\sigma}$ is also a with-safe-with-theft scenario, the inequality is strict, i.e., $C \succ C^U_{\overline{\sigma}}$.

We claim that

$$C^A_{\overline{\sigma}} \succeq C^U_{\overline{\sigma}}. \tag{5}$$

This is because, if instead $C^U_{\overline{\sigma}} \succ C^A_{\overline{\sigma}}$, then because $C \succ C^U_{\overline{\sigma}}$ and TKR, we get $C \not\subseteq C^A_{\overline{\sigma}}$, which is false because $C \subseteq C^A_{\overline{\sigma}}$.

By definition, for any pair of complement scenarios, we have $C^U_\sigma = C^A_{\overline{\sigma}}$ and $C^U_{\overline{\sigma}} = C^A_\sigma$. Recasting eq. (5), we get $C^U_\sigma \succeq C^A_\sigma$, which contradicts eq. (4). □

Next, we prove that the user can win in exactly half of the with-safe-with-theft scenarios Σ_{wswt}.

Lemma 7. *Any well-formed mechanism succeeds in either a with-safe-with-theft scenario $\sigma \in \Sigma_{\mathsf{wswt}}$ or its complement $\overline{\sigma}$, i.e.,* $\mathit{Suc}(M_{\mathsf{wf}}, \sigma) \vee \mathit{Suc}(M_{\mathsf{wf}}, \overline{\sigma})$.

Proof. By Theorem 3, for each scenario $\sigma \in \Sigma_{\mathsf{wswt}}$, its complement $\overline{\sigma}$ satisfies $\overline{\sigma} \neq \sigma$ and $\overline{\sigma} \in \Sigma_{\mathsf{wswt}}$.

Consider a pair of complement scenarios $\sigma, \overline{\sigma} \in \Sigma_{\mathsf{wswt}}$. Without loss of generality, assume that the mechanism M_{wf} is unsuccessful in σ, i.e., there exists a winning strategy S^A for the attacker that allows it to win an execution in σ.

By Lemma 5, the existence of a winning strategy S^A implies that the submit-early strategy S^A_{all} is also a winning strategy. And by Lemma 6, the messaging component of S^A_{all} is a winning strategy for the user in the complement scenario $\overline{\sigma}$. □

We conclude the proof of Theorem 4 by proving that the profile size of a well-formed mechanism is $P(n)$.

Lemma 8. *The profile size of any well-formed n-credential mechanism is $P(n)$.*

c_1 \ c_2	Theft	Leak	Lost	Safe
Theft	0	0	0	0
Leak	0	0	0	1
Lost	0	0	0	1
Safe	1	1	1	1

TABLE II: The profile of $M_{\mathsf{pr}}^{[c_1,c_2]}$.

Proof. Lemma 4 shows that a well-formed mechanism succeeds in all with-safe-no-theft scenarios ($\Sigma_{\mathsf{wsnt}} = \Sigma_{\mathsf{ws}} \cap \Sigma_{\mathsf{nt}}$). By basic set theory, $|\Sigma_{\mathsf{wsnt}}| = |\Sigma_{\mathsf{nt}}| - |\Sigma_{\mathsf{nt}} \setminus \Sigma_{\mathsf{ws}}|$ and $|\Sigma_{\mathsf{nt}} \setminus \Sigma_{\mathsf{ws}}| = |\Sigma_{\mathsf{nt}} \cap \overline{\Sigma_{\mathsf{ws}}}| = |\Sigma_{\mathsf{nt}} \cap \Sigma_{\mathsf{ns}}|$. Finally, since $|\Sigma_{\mathsf{nt}}| = 3^n$ and $|\Sigma_{\mathsf{nt}} \cap \Sigma_{\mathsf{ns}}| = 2^n$, we have $|\Sigma_{\mathsf{wsnt}}| = 3^n - 2^n$.

Lemma 7 implies that a well-formed mechanism succeeds in exactly half of the with-safe-with-theft scenarios ($\Sigma_{\mathsf{wswt}} = \Sigma_{\mathsf{ws}} \cap \Sigma_{\mathsf{wt}}$). From the proof of Theorem 3, $|\Sigma_{\mathsf{wswt}}| = (4^n - 2 \cdot 3^n + 2^n)$.

In total, a well-formed mechanism succeeds in $|\Sigma_{\mathsf{wsnt}}| + |\Sigma_{\mathsf{wswt}}|/2 = (4^n - 2^n)/2 = P(n)$ scenarios. □

C. Algorithms for maximal mechanisms

We now specify three algorithms that produce maximal n-credential mechanisms for any $n \geq 2$.

1) Priority mechanisms: The function J_{pr}^V is specified in Algorithm 1. The vector V defines an ordering over all the credentials, and J_{pr}^V is the priority judging function using V. The priority function selects the party that submits a unique high-priority credential.

Algorithm 1 The priority judging function J_{pr}^V

Require: V is a permutation over the elements of the set $\mathcal{C}_{\mathsf{all}}$.
function $J_{\mathsf{pr}}^V(C_0, C_1)$ ▷ $C_0 \subseteq \mathcal{C}_{\mathsf{all}}, C_1 \subseteq \mathcal{C}_{\mathsf{all}}$
  for $c = V_1, V_2, \ldots, V_n$ **do**
    if $c \in C_0 \wedge c \notin C_1$ **then return** id_0 ▷ $C_0 \succ C_1$
    if $c \in C_1 \wedge c \notin C_0$ **then return** id_1 ▷ $C_1 \succ C_0$
  return id_0 ▷ Default

Given a judging function J, we need to apply the translation procedure given in §IV-B to arrive at an authentication mechanism. We denote the corresponding priority mechanism by M_{pr}^V. For example, the 2-credential priority mechanism $M_{\mathsf{pr}}^{[c_1,c_2]}$ is in Fig. 2. Its profile is in Table II.

Lemma 9. *Given a permutation V over elements of the set $\mathcal{C}_{\mathsf{all}}$, the priority judging function J_{pr}^V is well-formed.*

Proof. We prove each of the three properties.

IA (ID-agnostic): To satisfy IA, we need to show that $(J_{\mathsf{pr}}^V(C_0, C_1) = \mathsf{id}_0) \Leftrightarrow (J_{\mathsf{pr}}^V(C_1, C_0) = \mathsf{id}_1)$.

Remove all the common credentials from both to get $C_0' \leftarrow C_0 \setminus C_1$ and $C_1' \leftarrow C_1 \setminus C_0$. Denote the highest priority credential in $C_0' \cup C_1'$ by c. Note that either $c \in C_0'$ or $c \in C_1'$ but both cannot be true. If $J_{\mathsf{pr}}^V(C_0, C_1) = \mathsf{id}_0$, then it implies that $c \in C_0' \subseteq C_0$, and therefore $J_{\mathsf{pr}}^V(C_1, C_0) = \mathsf{id}_1$. The other direction can be proven similarly.

KR (knowledge-rewarding): To satisfy KR, we need to show that if $C_0 \subset C_1$ then $C_0 \prec C_1$. We prove by contradiction, i.e., assume $C_0 \succ C_1$.

Like before, $C_0 \succ C_1$ implies that there exists a credential $c \in C_0$ such that $c \notin C_1$. But this contradicts $C_0 \subset C_1$.

TKR (transitive-knowledge): To satisfy TKR, we want to show that given three different credential sets C_0, C_1 and C_2, if $C_0 \succ C_1$ and $C_1 \succ C_2$ then $C_0 \not\subseteq C_2$. We prove by contradiction, i.e., assume $C_0 \subseteq C_2$. Since $C_0 \neq C_2$ by definition, the assumption really is $C_0 \subset C_2$.

Like before, remove the common credentials in C_0 and C_1 to derive $C_0' \leftarrow C_0 \setminus C_1$ and $C_1' \leftarrow C_1 \setminus C_0$. Since $C_0 \succ C_1$, the highest priority credential c across C_0' and C_1' belongs to C_0', i.e., $c \in C_0' \subseteq C_0$, $c \notin C_1$ and $\{c\} \succ C_1'$.

Note that since $C_0 \subset C_2$, the credential c is also present in C_2.

We now prove that $C_2 \succ C_1$ thereby contradicting the starting assumption. Repeat the previous exercise for the sets C_1 and C_2: let C_1'' and C_2'' be the reduced sets after all the common credentials are removed, i.e., $C_1'' \leftarrow C_1 \setminus C_2$ and $C_2'' \leftarrow C_2 \setminus C_1$. Note that the credential c from before satisfies $c \in C_2''$ because $c \notin C_1$ and $c \in C_0 \subset C_2$.

Also $C_0 \subset C_2$ implies $C_1'' \subseteq C_1'$ because $C_1 \setminus C_2 \subseteq C_1 \setminus C_0$. So $\{c\} \succ C_1''$ holds because $\{c\} \succ C_1'$ and $C_1'' \subseteq C_1'$, i.e., the credential c has higher priority than any credential in C_1' and consequently also in C_1''.

And finally, because $c \in C_2''$, we have $C_2'' \succ C_1''$ and $C_2 \succ C_1$. This contradicts the property's assumption that $C_1 \succ C_2$.

Therefore, priority judging functions satisfy IA, KR and TKR, and hence they are well-formed. □

2) Priority with exception: Mechanisms in this category are similar to priority mechanisms except when the last two credentials in the priority rule are submitted by the two parties. For example, if the priority vector is $V = [c_2, c_3, c_1]$, then the exception is $\{c_1\} \succ \{c_3\}$. The resultant judging function is denoted by J_{pre}^V and is specified in Algorithm 2.

Lemma 10. *Given a permutation V over elements of the set $\mathcal{C}_{\mathsf{all}}$, the priority with exception judging function J_{pre}^V is well-formed.*

Proof. It is straightforward to see that the judging function satisfies the first two properties: IA (ID-agnostic) and KR (knowledge-rewarding).

We now prove TKR. To satisfy TKR, we want to show that given three different credential sets C_0, C_1 and C_2, if $C_0 \succ C_1$ and $C_1 \succ C_2$ then $C_0 \not\subseteq C_2$.

Since $C_0 \succ C_1$, we have two cases: (A) $C_0 \succ C_1$ is not the exception or (B) $C_0 \succ C_1$ is the exception. Similarly $C_1 \succ C_2$ means: (I) $C_1 \succ C_2$ is not the exception or (II) $C_1 \succ C_2$ is the exception. One of the four cases: A-I, B-I, A-II and B-II must be true. We consider each one separately.

The proof for the case A-I (no exceptions) is exactly the same as the one in Lemma 9.

Algorithm 2 Priority with exception J^V_{pre}

Require: V is a permutation over the elements of the set $\mathcal{C}_{\mathsf{all}}$.
function $J^V_{\mathsf{pre}}(C_0, C_1)$ ▷ $C_0 \subseteq \mathcal{C}_{\mathsf{all}}, C_1 \subseteq \mathcal{C}_{\mathsf{all}}$
  if $C_0 = V_{n-1} \wedge C_1 = V_n$ **then return** id_1
  if $C_0 = V_n \wedge C_1 = V_{n-1}$ **then return** id_0
  for $c = V_1, V_2, \ldots, V_n$ **do**
    if $c \in C_0 \wedge c \notin C_1$ **then return** id_0 ▷ $C_0 \succ C_1$
    if $c \in C_1 \wedge c \notin C_0$ **then return** id_1 ▷ $C_1 \succ C_0$
  return id_0 ▷ Default

c_1 safe

c_2 \ c_3	T	Le	Lo	S
T	1	1	1	1
Le	1	1	1	1
Lo	1	1	1	1
S	1	1	1	1

c_1 lost

c_2 \ c_3	T	Le	Lo	S
T	0	0	0	0
Le	0	0	0	1
Lo	0	0	0	1
S	1	1	1	1

c_1 leaked

c_2 \ c_3	T	Le	Lo	S
T	0	0	0	0
Le	0	0	0	1
Lo	0	0	0	1
S	1	1	1	1

c_1 theft

c_2 \ c_3	T	Le	Lo	S
T	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	0
S	0	0	0	0

TABLE III: The profile of $M^{[c_1,c_2,c_3]}_{\mathsf{pr}}$.

Case B-II (both exceptions) is actually impossible because there is only one exception and the three credential sets are different.

The case B-I where $C_0 \succ C_1$ is the exception can also be ruled out due to the way we define an exception, namely, that the credential declared worse by the exception is the worst non-empty set of credentials. For example, if the priority vector is $V = [c_2, c_3, c_1]$, then case B-I corresponds to $C_0 = \{c_1\}$ and $C_1 = \{c_3\}$. But no non-empty set C_2 exists s.t. $C_1 \succ C_2$. So $C_2 = \phi$ and therefore $C_0 \not\subseteq C_2$.

The remaining case is A-II where $C_1 \succ C_2$ is the exception. We prove this by contradiction, i.e., assume $C_0 \subseteq C_2$ or to be precise $C_0 \subset C_2$ because $C_0 \neq C_2$. But no non-empty C_0 exists because the only possible C_0 satisfying $C_0 \subset C_2$ is $C_0 = \phi$ and it does not satisfy $C_0 \succ C_1$. This concludes the proof since we ruled out all the four cases. □

3) Majority mechanisms: Mechanisms in this category favor the party submitting the *most credentials*, so $C \succ C'$ if $|C| > |C'|$. To break ties, different strategies are possible which we model through a tie-breaking function $T(C_0, C_1) \mapsto \{\mathsf{id}_0, \mathsf{id}_1\}$. We only consider ID-agnostic tie-breaking functions, i.e., if $\mathsf{id}_0 \leftarrow T(C_0, C_1)$ then $\mathsf{id}_1 \leftarrow T(C_1, C_0)$. The resultant judging function is denoted J^T_{maj} and is specified in Algorithm 3. The profile of a 3-credential majority mechanism is shown in Table IV. For comparison, we also show the profile of the 3-credential priority mechanism in Table III.

c_1 safe

c_2 \ c_3	T	Le	Lo	S
T	0	0	0	1
Le	0	1	1	1
Lo	0	1	1	1
S	1	1	1	1

c_1 lost

c_2 \ c_3	T	Le	Lo	S
T	0	0	0	1
Le	0	0	0	1
Lo	0	0	0	1
S	0	1	1	1

c_1 leaked

c_2 \ c_3	T	Le	Lo	S
T	0	0	0	1
Le	0	0	0	1
Lo	0	0	0	1
S	0	1	1	1

c_1 theft

c_2 \ c_3	T	Le	Lo	S
T	0	0	0	0
Le	0	0	0	1
Lo	0	0	0	1
S	0	1	1	1

TABLE IV: The profile of the 3-credential majority mechanism with the $[c_3, c_2, c_1]$ priority tie-breaking function.

Algorithm 3 The majority judging function J^T_{maj}

function $J^T_{\mathsf{maj}}(C_0, C_1)$ ▷ $C_0 \subseteq \mathcal{C}_{\mathsf{all}}, C_1 \subseteq \mathcal{C}_{\mathsf{all}}$
  if $|C_0| > |C_1|$ **then return** id_0
  if $|C_1| > |C_0|$ **then return** id_1
  return $T(C_0, C_1)$ ▷ If $|C_0| = |C_1|$

Lemma 11. *Given any ID-agnostic tie-breaking function T, the majority judging function J^T_{maj} is well-formed.*

Proof. We prove the three properties.

IA (ID-agnostic): To satisfy IA, we need to show that $(J^T_{\mathsf{maj}}(C_0, C_1) = \mathsf{id}_0) \Leftrightarrow (J^T_{\mathsf{maj}}(C_1, C_0) = \mathsf{id}_1)$.

Assume $|C_0| \neq |C_1|$. If $J^T_{\mathsf{maj}}(C_0, C_1) = \mathsf{id}_0$, then it must be that $|C_0| > |C_1|$ (Algorithm 3) and hence $J^T_{\mathsf{maj}}(C_1, C_0) = \mathsf{id}_1$. The other direction can be proven similarly.

Now assume $|C_0| = |C_1|$. Recall that the tie-breaker T is ID-agnostic, i.e., $(T(C_0, C_1) = \mathsf{id}_0) \Leftrightarrow (T(C_1, C_0) = \mathsf{id}_1)$. $|C_0| = |C_1|$ implies that $J^T_{\mathsf{maj}}(C_0, C_1) = T(C_0, C_1)$, which completes the proof.

KR (knowledge-rewarding): To satisfy KR, we need to show that if $C_0 \subset C_1$ then $C_0 \prec C_1$. It follows from Algorithm 3 because $C_0 \subset C_1$ implies $|C_0| < |C_1|$, and therefore $C_0 \prec C_1$.

TKR (transitive-knowledge): Given three different credential sets C_0, C_1 and C_2 s.t. $C_0 \succ C_1$, and $C_1 \succ C_2$, we want to show that $C_0 \not\subseteq C_2$ to satisfy TKR.

For any majority-based judging function, if $C \succ C'$ then $|C| \geq |C'|$. So we have $|C_0| \geq |C_1| \geq |C_2|$.

So either $|C_0| > |C_2|$ or $|C_0| = |C_2|$ is true. If $|C_0| > |C_2|$ then $C_0 \not\subseteq C_2$ and we are done.

It remains to prove for $|C_0| = |C_2|$. But in this case $C_0 \not\subseteq C_2$ follows as $C_0 \neq C_2$ (by definition) and $|C_0| = |C_2|$. □

Number of maximal mechanisms: We now count the total number of priority and majority functions for a given n.

The number of priority mechanisms is equal to the number of different permutations over n elements, $n!$. All of them are equivalent to each other. The same is true for priority with exception mechanisms.

The number of majority mechanisms is equal to the number of ways of resolving ties. A tie occurs when $C_0 \neq C_1$ but $|C_0| = |C_1|$. Let $|C_0| = |C_1| = s$ where $1 \leq s < n$ (note that s can't be equal to n since $C_0 \neq C_1$). For a given s, there are

$$f(n,s) = \frac{\binom{n}{s}(\binom{n}{s} - 1))}{2},$$

possible tie situations. This is because, there are $\binom{n}{s}$ ways of picking an s-sized set C_0, and $\binom{n}{s} - 1$ ways of picking another set $C_1 \neq C_0$. We divide by two as each tie situation repeats twice. Summing over all s, the total number of distinct tie situations is

$$q(n) = \sum_{s=1}^{n-1} f(n,s). \tag{6}$$

A tie-breaking function can decide each situation in two ways: pick C_0 or C_1. So there are $2^{q(n)}$ different tie-breaking functions, or in other words, $2^{q(n)}$ different majority functions. Some of the resulting majority mechanisms are equivalent to each other, but we're unaware of any general rule.

All of the above mechanisms are well-formed due to Lemma 9, Lemma 10 and Lemma 11, and the corresponding mechanisms are maximal due to Theorem 4.

V. Complete maximal sets

A *complete maximal mechanism set* is a minimal set of n-credential mechanisms such that *any other n-credential mechanism is either equivalent to or worse than some mechanism in this set.*

Definition 17 (Complete maximal mechanism set)**.** *A complete maximal mechanism set of n-credential mechanisms $\mathcal{O}_n = \{M_1, M_2 \ldots\}$ satisfies three properties: (1) each $M \in \mathcal{O}_n$ satisfies $M \in \mathcal{M}_n$ and is maximal; (2) any two distinct members are incomparable, i.e., $\forall M, M' \in \mathcal{O}_n, M \neq M' : M \nsim M'$; (3) any n-credential mechanism is worse or equivalent to some mechanism in $\mathcal{O}$, i.e., $\forall M \in \mathcal{M}_n : \exists M' \in \mathcal{O}$ s.t. $M \preceq M'$.*

Note that for a given n, several different complete maximal sets can exist. But their sizes will be the same as each mechanism in one of the maximal sets will have an equivalent mechanism in the other. We now prove the same.

Lemma 12. *For all n, the size of all n-credential complete maximal sets is the same.*

Proof. Consider two different complete maximal sets $\mathcal{O}_1$ and $\mathcal{O}_2$. Each mechanism $M_2 \in \mathcal{O}_2$ must be equivalent or worse to a mechanism $M_1 \in \mathcal{O}_1$ (Definition 17). But all mechanisms in a complete maximal set are maximal, therefore it must be that $M_2 \cong M_1$. This process can be repeated for all mechanisms in $\mathcal{O}_2$. So, for each mechanism in $\mathcal{O}_2$, there exists an equivalent mechanism in the other set $\mathcal{O}_1$, and vice versa. And because no two mechanisms within $\mathcal{O}_1$ (or $\mathcal{O}_2$) can be equivalent, there exists a one-to-one mapping between $\mathcal{O}_1$ and $\mathcal{O}_2$. So $|\mathcal{O}_1| = |\mathcal{O}_2|$. □

We now present complete maximal sets for all $n \leq 3$. These sets are composed of bounded-delay mechanisms introduced in the previous section.

One-credential mechanisms: The case of one-credential mechanisms is not very interesting because Theorem 2 implies that three of the four possible scenarios are always won by the attacker. The remaining scenario is the credential c_1 being safe. The simple judging function J satisfying $\{c_1\} \succ_J \phi$ is well-formed, and the corresponding mechanism is the only one in the complete maximal set $\mathcal{O}_1$.

A. Two-credential mechanisms

The complete maximal set $\mathcal{O}_2$ contains just the priority mechanism defined using the ordering $[c_1, c_2]$. (This can also be viewed as majority mechanisms that use priority rule to break ties.)

Lemma 13. *The profile matrix of $M_{\mathsf{pr}}^{[c_1,c_2]}$ is as shown in Table II.*

Proof. The 3x3 grid with no safe scenarios are unsuccessful, i.e., won by the attacker due to Theorem 2.

Since the priority judging function is well-formed, $M_{\mathsf{pr}}^{[c_1,c_2]}$ is maximal (Lemma 9). Theorem 4 says that all maximal mechanisms are secure in with-safe-no-theft scenarios $\Sigma_{\mathsf{wsnt}} = \Sigma_{\mathsf{ws}} \cap \Sigma_{\mathsf{nt}}$ and half of the with-safe-with-theft scenarios ($\Sigma_{\mathsf{wswt}} = \Sigma_{\mathsf{ws}} \cap \Sigma_{\mathsf{wt}}$).

This fixes the values in all scenarios of the matrix except two: the scenario $\sigma = \{\mathsf{safe}, \mathsf{theft}\}$ and its complement $\overline{\sigma} = \{\mathsf{theft}, \mathsf{safe}\}$. Theorem 4 says that at most one of $\sigma, \overline{\sigma}$ is secure; we need to find out which. The judging function $J_{\mathsf{pr}}^{[c_1,c_2]}$ favors the user in the scenario σ and the attacker in $\overline{\sigma}$. □

Theorem 5. *The complete maximal set of 2-credential mechanisms is $\mathcal{O}_2 = \{M_{\mathsf{pr}}^{[c_1,c_2]}\}$.*

Proof. We prove $\mathcal{O}_2$ satisfies the three requirements in Definition 17.

Req. 1: The first requirement is that $M_{\mathsf{pr}}^{[c_1,c_2]}$ is maximal. This directly follows from Lemma 9 (priority judging functions are well-formed) and Theorem 4 (well-formed functions produce maximal mechanisms).

Req. 2: The second requirement states that every pair of mechanisms in $\mathcal{O}_2$ must be incomparable to each other. This is trivially satisfied because $\mathcal{O}_2$ has only one mechanism.

Req. 3: The third requirement states that any 2-credential mechanism M must satisfy $M \preceq M_{\mathsf{pr}}^{[c_1,c_2]}$. Recall that $M \preceq M_{\mathsf{pr}}^{[c_1,c_2]}$ implies the existence of a mechanism $M' \cong M_{\mathsf{pr}}^{[c_1,c_2]}$ such that $\mathit{prof}(M) \subseteq \mathit{prof}(M')$. There are only two choices for M' that yield distinct profiles because $\mathit{expand}(M_{\mathsf{pr}}^{[c_1,c_2]}) = \{M_{\mathsf{pr}}^{[c_1,c_2]}, M_{\mathsf{pr}}^{[c_2,c_1]}\}$. So we have $M' \in \{M_{\mathsf{pr}}^{[c_1,c_2]}, M_{\mathsf{pr}}^{[c_2,c_1]}\}$. And we are trying to prove that for any mechanism M, one of $\mathit{prof}(M) \subseteq \mathit{prof}(M_{\mathsf{pr}}^{[c_1,c_2]})$ or $\mathit{prof}(M) \subseteq \mathit{prof}(M_{\mathsf{pr}}^{[c_2,c_1]})$ is true.

We prove by contradiction. Assume there exists a mechanism M such that $\mathit{prof}(M) \not\subseteq \mathit{prof}(M_{\mathsf{pr}}^{[c_1,c_2]})$ and $\mathit{prof}(M) \not\subseteq \mathit{prof}(M_{\mathsf{pr}}^{[c_2,c_1]})$.

Consider the set of seven with-safe scenarios $\Sigma_{\mathsf{ws}} = \{\hat{\sigma}_1, \ldots, \hat{\sigma}_7\}$. W.l.o.g, let $\hat{\sigma}_1 = \{\mathsf{safe}, \mathsf{theft}\}$ (i.e., c_1 safe, c_2 theft) and $\hat{\sigma}_7 = \{\mathsf{theft}, \mathsf{safe}\}$ is the complement of $\hat{\sigma}_1$. Lemma 13 shows that $\mathit{prof}(M_{\mathsf{pr}}^{[c_1,c_2]}) = \{\hat{\sigma}_1, \ldots, \hat{\sigma}_6\}$ and $\mathit{prof}(M_{\mathsf{pr}}^{[c_2,c_1]}) = \{\hat{\sigma}_2, \ldots, \hat{\sigma}_7\}$.

By Theorem 2, the profile of M is a subset of Σ_{ws}, i.e., $\mathit{prof}(M) \subseteq \Sigma_{\mathsf{ws}}$. By basic set theory, the only way to guarantee $\mathit{prof}(M) \not\subseteq \mathit{prof}(M_{\mathsf{pr}}^{[c_1,c_2]})$ and $\mathit{prof}(M) \not\subseteq \mathit{prof}(M_{\mathsf{pr}}^{[c_2,c_1]})$ is if $\{\hat{\sigma}_1, \hat{\sigma}_7\} \in \mathit{prof}(M)$. But Theorem 1 rules this out as they are complement scenarios. □

B. Three-credential mechanisms

The complete maximal set $\mathcal{O}_3$ can be grouped into two disjoint sets: *majority* and *priority* denoted by $\mathcal{O}_{\mathsf{maj},3}$ and $\mathcal{O}_{\mathsf{pr},3}$ respectively, so $\mathcal{O}_3 = \mathcal{O}_{\mathsf{maj},3} \cup \mathcal{O}_{\mathsf{pr},3}$.

By eq. (6), $q(3) = 64$, i.e., there are 64 majority-based mechanisms in total, of which only 12 mechanisms are distinct. The profile of a majority mechanism $M_{\mathsf{maj}}^{[c_1,c_2,c_3]}$, i.e., using priority vector $[c_1, c_2, c_3]$ to break ties, is shown in Table IV.

There are two priority-based mechanisms: simple priority mechanism specified by Algorithm 1 and priority with an exception specified by Algorithm 2. We have $\mathcal{O}_{\mathsf{pr},3} = \{M_{\mathsf{pr}}^{[c_1,c_2,c_3]}, M_{\mathsf{pre}}^{[c_1,c_2,c_3]}\}$.

In summary, $\mathcal{O}_3$ is made up of two priority-based mechanisms and twelve majority-based ones.

Theorem 6. *The complete maximal set of 3-credential mechanisms is* $\mathcal{O}_3 = \mathcal{O}_{\mathsf{pr},3} \cup \mathcal{O}_{\mathsf{maj},3}$.

Proving the first two requirements of Definition 17 is straightforward. To prove the final one, we use a constraint solver to find the profile of a hypothetical mechanism M satisfying two types of constraints. The first type encodes that M must be incomparable with any mechanism in $\mathcal{O}_3$. The second type relies on the following observation: if M fails in a scenario σ, then it must fail in all scenarios $\widehat{\sigma}$ that are worse than σ. $\widehat{\sigma}$ is *worse* than σ if the attacker knows (equal or) more credentials whereas the user knows (equal or) fewer credentials. More details are in §A due to lack of space.

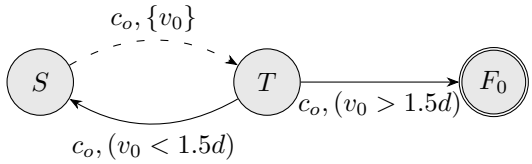
C. Larger number of credentials

We do not present the complete maximal sets for n-credential mechanisms with $n \geq 4$. This is because it is not clear if newer types of exceptions arise when generalizing PRE-based mechanisms to $n \geq 4$ credentials.

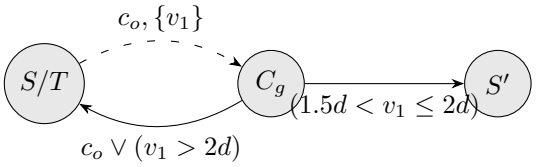
We are also unable to find a meaningful lower bound for $|\mathcal{O}_n|$. This problem is hard because it is not clear how to identify equivalences among majority mechanisms when $n \geq 4$. However, we do present a slightly weaker result: a lower bound on the isomorphic expansion of $\mathcal{O}_n$.

Define an *expanded maximal mechanism set* as a union of isomorphic expansion of all the mechanisms in the complete maximal set, i.e., $\mathcal{O}_n^{\mathsf{iso}} = \bigcup_{M \in \mathcal{O}_n} \mathit{expand}(M)$. Below is a lower bound on its size.

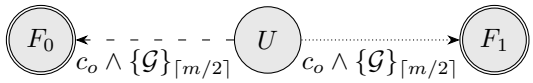
Theorem 7. *The size of the expanded complete maximal set of* n*-credentials for* $n \geq 4$ *satisfies* $|\mathcal{O}_n^{\mathsf{iso}}| \geq q(n) + 2n!$



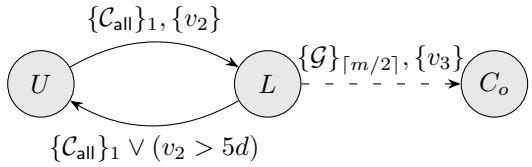
(a) Transfer to an unapproved address



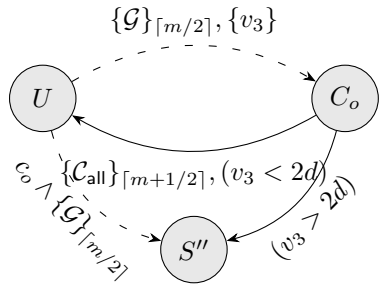
(b) Change guardians



(c) Immediately transfer to an unapproved address



(d) Locking wallet



(e) Change owner

Fig. 3: Argent mechanism sub-automata

Proof. Lemma 11 shows that $q(n)$ majority n-credential maximal mechanisms exist and Lemma 9, Lemma 10 show that there are $2n!$ priority-based maximal mechanisms. The lower bound on the size of $\mathcal{O}_n^{\mathsf{iso}}$ follows. □

VI. Applications

We now apply our framework to analyze a cryptocurrency wallet (§VI-A), an online bank website (§VI-B), and some prior work (§VI-C).

A. Argent cryptocurrency wallet

A popular approach to design non-custodial cryptocurrency wallets is social recovery, where a user's social circle, e.g.,

friends and family, is used to manage keys. It is considered as one of the best approaches due to its ease-of-use [23]. We analyze Argent [4], a popular social recovery wallet used by millions of users [16].

The Argent mechanism uses $m+1$ credentials, consisting of an owner credential c_o and a set of m so-called *guardian* credentials $\mathcal{G} = \{g_1, g_2, \ldots, g_m\}$. The owner credential is a cryptographic key on the user's mobile phone. Guardians can be anything from a friend's cryptocurrency wallet to a hardware wallet or a paid third-party service. We thus have $\mathcal{C}_{\mathsf{all}} = \{c_o, g_1, g_2, \ldots, g_m\}$.

Each owner maintains a list of *approved addresses* (they call it *trusted contacts*) to which it can send funds immediately. Transferring funds to a unapproved address requires additional steps. The Argent wallet provides capabilities to add or remove approved addresses, add or remove guardians, change owner (if phone is lost or stolen) and transfer deposited funds.

To analyze the mechanism, we review all the attacker's options to withdraw funds into an unapproved address. To simplify the presentation, we first review different paths separately. Figure 3 depicts five automata, each capturing a different functionality of the Argent mechanism.

Recall that dashed (resp., dotted) edges indicates an edge that can be taken only by player id_0 (resp., id_1). Solid edges can be taken by either player. The state S is the starting state of the automaton. Two final states exist: F_0 (resp., F_1) stands for the state in which player zero (resp., player one) wins. We only show half of the automaton, namely the portion assuming the first move is made by player id_0; we omit the other half because it is symmetric. Note that reaching a final state signifies a successful withdrawal.

If credentials are unchanged, there are only two ways for the attacker to withdraw funds to an address it controls (Fig. 3a, Fig. 3c). She can get an approval to this address and then withdraw, a process we call *slow withdrawal*. In order to do so, the owner needs to initiate an action (S to T). The addition becomes effective after 1.5 days, at which point the owner can withdraw (T to F_0). The approval process can be cancelled during the 1.5 day period (T to S): this is useful if the owner credential c_o is leaked.

The attacker can also try to withdraw funds by first changing the set of credentials. One way is to change the set of guardians (Fig. 3b). The owner initiates the process (S to C_g) and can be finalized only during a time window that starts 1.5 days after initiation and lasts for 12 hrs thereafter. As in Fig. 3a, the guardian change process can be cancelled by c_o.

Note that a guardian change implies a change in the underlying scenario. The execution moves to another copy of the same automaton, indicated by the new start state S', with a new scenario σ' that reflects the change to guardians. For example, if player id_0 is able to add a guardian, then the new scenario σ' has a new guardian credential g_{m+1} that is known to id_0 alone, and $\mathcal{C}'_{\mathsf{all}} = \mathcal{C}_{\mathsf{all}} \cup \{g_{m+1}\}$.

Another way to withdraw funds is to order a *fast withdrawal* with the owner credential and at least half of the guardian credentials (Fig. 3c). The state U represents all unlocked states, i.e., S or T or C_g.

g_1 \ c_o	Theft	Leak	Lost
Safe	0	1	1

c_o \ g_1	Theft	Leak	Lost
Safe	0	0	1

TABLE V: Profile of Argent with 1 guardian.

Argent implements *locking*, a feature intended for situations when the owner suspects a compromise [4]. As shown in Fig. 3d, the mechanism can be moved from any state into a locked state by showing any one credential. A limited set of actions are possible in the locked state, namely, change owner and unlock. Unlocking the automaton, i.e., moving back into its previous state can be done by showing any credential.

Finally, Argent allows to change the owner, a process they call *recovery*. There are two ways to do so (Fig. 3e). If the current owner credential is lost, the automaton can be moved from any unlocked state to the recovery state C_o with $\lceil m/2 \rceil$ guardian credentials. The new owner address is specified in this step. Finalizing this new owner takes 2 days. In this period, owner change can be cancelled by showing $\lceil (m+1)/2 \rceil$ credentials, which can include the original owner: this is useful if the owner credential was not actually lost.

The second way is for when the owner credential is safe, e.g., if the user wants to transfer the Argent app between phones. In this case, there is no need to wait. The owner can be changed immediately by showing the owner credential c_o and $\lceil m/2 \rceil$ guardians.

Similar to when a guardian change happens, a change of owner moves the execution back to a start state S'' with a modification to the scenario. For example, if player id_0 is able to change the owner, then the new scenario will reflect that: the new credential c_o is known to id_0 alone.

Note that we do not model initiating many withdrawals or changing multiple guardians concurrently for simplicity.

Profile analysis: We now analyze the security of Argent's mechanism with one and with two guardians ($m = 1, 2$).

The one guardian case has just two credentials: an owner c_o and a guardian g_1. The profile is in Table V, with all the scenarios containing exactly one safe credential.

The user can reach a final state in two ways: (a) submit both the credentials (Fig. 3c) or (b) submit just c_o and wait for 1.5 days (Fig. 3a). So if one of the credentials is lost but the other is safe, then the user's winning strategy is to revoke the lost credential. This is winning because all the credentials will be safe in the new automaton. We consider other scenarios below.

g_1 safe, c_o leaked: In this scenario, the user wins. The winning strategy is to change the owner by going to C_o and then S''. Note that irrespective of what the attacker does, the user will be able to reach C_o; in particular, even if the attacker locks the mechanism (reaches L), there is an edge from L to C_o.

g_1 safe, c_o theft: Now the mechanism fails because the attacker can lock the mechanism and the user cannot use the edge from L to C_o. Although the user can unlock it (L to U), the

attacker can immediately lock the mechanism again; that is, the mechanism is stuck in an infinite loop.

c_o *safe:* If g_1 is theft or leak, then the attacker's winning strategy is very similar to above except to move the automaton to the owner change state C_o whenever it moves out of it.

In summary, the Argent 1 guardian mechanism's profile has four scenarios, and is therefore worse than our maximal mechanism (Table II). We similarly analyze the two guardian case in §B.

Using our maximal mechanisms: We now discuss how to use our maximal mechanisms to achieve the same functionality as Argent. First, we omit the locking functionality provided by Argent as it does not improve security in our model. Rather than using different mechanisms for the various functions, e.g., change owner / guardian, we propose the use of a maximal mechanism for all functionalities, thus vastly simplifying the design. We propose the use of the majority mechanism due to its simplicity.

One missing feature from our proposed design is fast withdrawals (Fig. 3c). However, it is easy to add it: simply move to the final state if *all the credentials* are submitted. The modified mechanism is still maximal, i.e., doesn't break PA, KR and TKR, since the fast path can only be enabled if all the credentials are input. However, it does incur a usability hit: for all $m > 1$, this requires the user to do more work in gathering guardian approvals than the Argent's mechanism. (If $m = 1$, the user needs to submit all the credentials anyway.)

B. Bank account

We model the authentication mechanisms used by HDFC bank [24]. The bank provides a traditional web portal for the users to login with their password credential c_p. We assume that 2FA is enabled, in particular, one time PINs are received on the registered mobile number; this credential is denoted c_m. The bank allows users to add additional factors if needed, e.g., email, but we do not model it for simplicity. Finally, users can change the registered mobile number by submitting a request at the bank along with an ID proof [25]. We model any acceptable identity proof using the credential c_{id}. There are other ways to change the mobile number [26], e.g., use a debit card and visit an ATM, but again, we do not model it for simplicity.

Like before, we focus on how to withdraw money to a new (attacker-controlled) account number, modeled in the first automaton in Fig. 4. It involves using the password c_p to login and authenticating via mobile c_m to initiate third the party addition process (S to A). This process takes 30 minutes [5], during which it can be cancelled by logging in to the bank portal (A to S). After the time elapses, money can be transferred to the new account. This requires another 2FA (A to F_0).

The bank allows changing the password after authenticating via mobile, modeled in the second automaton. In order to change the registered mobile number, the user must submit a form along with relevant identity proof c_{id}. We could not find how long this process takes in their online documentation, but based on other blogs [26], we set it to 3 days.

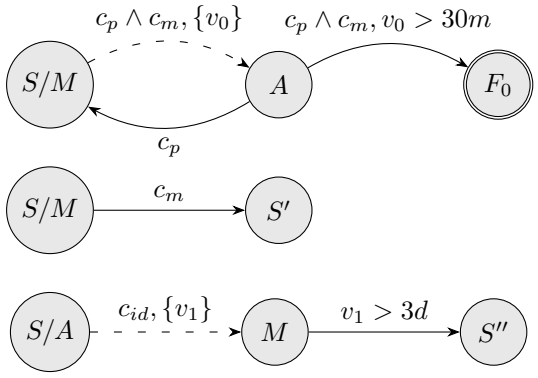


Fig. 4: HDFC bank sub-automata: send money to a new account, change password and change mobile number respectively.

c_p, c_{id} \ c_m	Theft	Leak	Lost
Safe	0	0	1

c_m, c_{id} \ c_p	Theft	Leak	Lost
Safe	1	1	1

c_m, c_{id} \ c_p	Theft	Leak	Lost	Safe
Theft	0	0	0	0
Leak	0	0	0	0
Lost	0	0	0	0
Safe	1	1	1	1

TABLE VI: Profile of the bank mechanism.

Profile analysis: We compute the profile assuming c_{id} is safe (Table VI) for simplicity. If c_m is safe, then the mechanism succeeds irrespective of the state of c_p. The user's winning strategy is to change the password. The mechanism also succeeds if c_p is safe and c_m is lost because c_{id} is safe. In other cases, the delay of one day in changing the mobile number is sufficient to withdraw money by the attacker. Like with Argent, this mechanism is also worse than our maximal mechanism.

We illustrate the redundance of the password credential c_p now. We find that the mechanism's success does not change irrespective of the state of c_p (shown in Table VI), hence c_p can be safely removed from the mechanism. While the redundance of passwords in some settings was observed by prior work [27], ours is the first to prove it in a rigorous manner to our knowledge.

C. Prior work

We depict several n-credential mechanisms where $n \leq 2$ together with their security profiles in Fig. 5. Each rectangle represents a mechanism with its profile in succinct form shown adjacently. Arrows between mechanisms show the $\succ$ relation, i.e., an arrow from mechanism M_1 to M_2 implies $M_2 \succ M_1$. Adjoining a node is the profile of the mechanism restricted

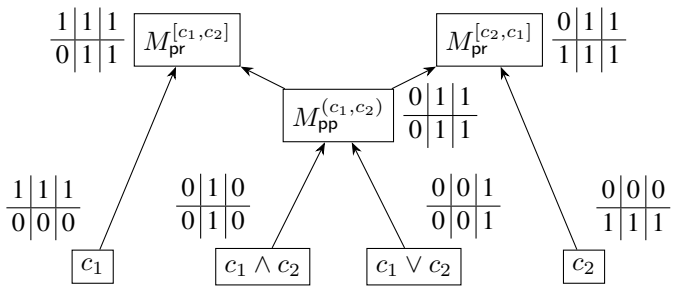


Fig. 5: One and two credential mechanisms.

to with-safe scenarios like before. The first row corresponds to c_1 being safe and c_2 being theft, leak or loss in that order, and the reverse for the second row.

The two maximal mechanisms, namely, $M_{\mathsf{pr}}^{[c_1,c_2]}$ and its isomorphism, take the top row in the lattice. The bottom row shows standard approaches, using a single credential, or joining two credentials c_1 and c_2 by either $\wedge$ or $\vee$.

The middle row represents a mechanism put forth by Zhang et al. [13], called Paralysis Proofs, which is the most secure mechanism from prior work to the best of our knowledge. We model their mechanism using our framework, denoted by $M_{\mathsf{pp}}^{(c_1,c_2)}$ (automata in). With two credentials, they use an AND-mechanism $c_1 \wedge c_2$ with an additional feature. Any credential can be challenged, and if no response is received within a fixed duration, then the challenged credential is removed. For example, if c_1 is successfully challenged, the new mechanism becomes just c_2. In this manner, they handle credential loss. As Fig. 5 illustrates, our maximal mechanisms achieve better security than the Paralysis Proofs mechanism.

VII. Communication-Channel Implementation

Our model assumes synchronous communication between the mechanism and the user. It is necessary not only to implement the logic of the mechanism, but also to ensure the synchrony of the communication.

As an example consider email messages users receive for adding a withdrawal address to a bank account (§VI-B). The email messages should serve as a synchronous channel. But if an attacker gains temporary access to a user's mobile, she can initiate the address addition and delete the email message, thus voiding the channel effectiveness. To overcome the above attack, we propose to make security notification emails (or messages) *sticky*, that is, they remain at the top of a user's inbox with a visible emphasis for the duration necessary to guarantee that the user sees them.

In the context of smart contracts on a public blockchain, the notification is the smart-contract state change, which is public; however the user will only observe it if her devices monitor the chain. And despite the robustness of the blockchain as a whole, a single monitoring device can be targeted by an attacker. Therefore, the user should monitor for sensitive events on multiple devices, as used for other goals, e.g., [28], [29], [30], [31].

VIII. Conclusion

We formalize the authentication problem in a synchronous environment and define the security profile for non-probabilistic evaluation of authentication mechanisms. After bounding profile size for any number of credentials n, we discover three types of mechanisms that achieve this bound and show they cover all maximal mechanisms for $n \leq 3$. These results can be directly harnessed to secure digital assets and improve the security of large-scale authentication systems – by ensuring the security of synchronous communication channels and by using our methodology to design maximal mechanisms.

References

[1] F. T. Commission, "Consumer sentinel network data book 2021," Tech. Rep., February 2022.

[2] Chainalysis, "60% of bitcoin is held long term as digital gold. what about the rest?" https://blog.chainalysis.com/reports/bitcoin-market-data-exchanges-trading/, June 2020.

[3] P. Jha, "The aftermath of axie infinity's $650m ronin bridge hack," *Cointelegraph*, 2022, https://cointelegraph.com/news/the-aftermath-of-axie-infinity-s-650m-ronin-bridge-hack.

[4] "Argent specification," https://github.com/argentlabs/argent-contracts/blob/develop/specifications/specifications.pdf, April 2021.

[5] "Cooling period: Get time to review newly added beneficiaries," https://www.hdfcbank.com/personal/useful-links/security/security-measures/cooling-period, 2022.

[6] "Taxpayer guide to identity theft," https://www.irs.gov/newsroom/taxpayer-guide-to-identity-theft, Internal Revenue Service, 2022.

[7] L. O'Gorman, "Comparing passwords, tokens, and biometrics for user authentication," *Proceedings of the IEEE*, 2003.

[8] C. Harper, "Multisignature wallets can keep your coins safer (if you use them right)," 2020, coindesk.

[9] E. V. Mangipudi, U. Desai, M. Minaei, M. Mondal, and A. Kate, "Uncovering impact of mental models towards adoption of multi-device crypto-wallets," Cryptology ePrint Archive, Paper 2022/075, 2022.

[10] T. Be'ery, "Threshold signatures: The future of private keys," https://zengo.com/threshold-signatures-the-future-of-private-keys/, 2018.

[11] I. Eyal, "On cryptocurrency wallet design," in *Tokenomics*, 2021.

[12] S. Hammann, S. Radomirović, R. Sasse, and D. Basin, "User account access graphs," in *ACM CCS*, 2019.

[13] F. Zhang, P. Daian, I. Bentov, I. Miers, and A. Juels, "Paralysis proofs: Secure dynamic access structures for cryptocurrency custody and more," in *AFT*, 2019.

[14] S. Blackshear, K. Chalkias, P. Chatzigiannis, R. Faizullabhoy, I. Khaburzaniya, E. K. Kogias, J. Lind, D. Wong, and T. Zakian, "Reactive key-loss protection in blockchains," Cryptology ePrint Archive, Report 2021/289, 2021.

[15] M. Möser, I. Eyal, and E. Gün Sirer, "Bitcoin covenants," in *FC*, 2016.

[16] E. Banulescu, "Argent wallet: Everything you need to know," July 2022, https://beincrypto.com/learn/argent-wallet/.

[17] C. Jacomme and S. Kremer, "An extensive formal analysis of multi-factor authentication protocols," *ACM TOPS*, 2021.

[18] M. Barbosa, A. Boldyreva, S. Chen, and B. Warinschi, "Provable security analysis of fido2," in *CRYPTO*, 2021.

[19] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "Sok: Research perspectives and challenges for bitcoin and cryptocurrencies," in *IEEE S&P*, 2015.

[20] J.-P. Aumasson, A. Hamelink, and O. Shlomovits, "A survey of ecdsa threshold signing," *Cryptology ePrint Archive*, 2020.

[21] P. Baratam, "Secure cryptocurrency exchange & wallet," https://www.coinvault.tech/wp-content/uploads/2020/10/CoinVault-Secure-Cryptocurrency-Exchange.pdf, 2020.

[22] S. Appelcline, "Using timelocks to protect digital assets," https://github.com/BlockchainCommons/SmartCustody/blob/master/Docs/Timelocks.md, 2021, [Accessed Sep 2022].

[23] V. Buterin, "Why we need wide adoption of social recovery wallets," https://vitalik.ca/general/2021/01/11/recovery.html, 2021.
[24] "HDFC bank," 2022, https://en.wikipedia.org/wiki/HDFC_Bank.
[25] "How to update contact details at a branch," 2022, https://www.hdfcbank.com/personal/useful-information/change-contact-details.
[26] "How to change mobile number in hdfc bank: 2 easy ways," 2022, https://thebankhelp.com/how-to-change-mobile-number-in-hdfc-bank/.
[27] S. Pearman, S. A. Zhang, L. Bauer, N. Christin, and L. F. Cranor, "Why people (don't) use password managers effectively," in *SOUPS*, 2019.
[28] P. McCorry, S. Bakshi, I. Bentov, S. Meiklejohn, and A. Miller, "Pisa: Arbitration outsourcing for state channels," in *AFT*, 2019.
[29] M. Khabbazian, T. Nadahalli, and R. Wattenhofer, "Outpost: A responsive lightweight watchtower," in *AFT*, 2019.
[30] "Ethereum push notification service," https://epns.io/, 2022.
[31] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "Sok: Layer-two blockchain protocols," in *FC*, 2020.

APPENDIX A
PROOF OF THEOREM 6

We present a few lemmas before proving Theorem 6. The first one says that ...

Lemma 14. *Given two scenarios σ, σ' using the same mechanism M and a user strategy S^U, attacker strategy S^A such that they can be employed in both the scenarios. Then $Win_\sigma(S^U, S^A) = Win_{\sigma'}(S^U, S^A)$.*

Proof. □

The next lemma says that, if the mechanism fails in a scenario σ, then it also fails in all scenarios $\widehat{\sigma}$ that are worse than σ. Being worse is defined as (a) $C^A_\sigma \subseteq \widehat{C^A_\sigma}$ and (b) $\widehat{C^U_\sigma} \subseteq C^U_\sigma$. $\widehat{\sigma}$ is *worse* than σ as the attacker knows (equal or) more credentials whereas the user knows (equal or) fewer credentials.

Formally, given a scenario σ, we define a worse scenario $\widehat{\sigma}$ through the following transform:

$$\begin{aligned}
\mathsf{safe} &\mapsto \mathsf{safe/leak/loss/theft}\\
\mathsf{theft} &\mapsto \mathsf{theft}\\
\mathsf{leak} &\mapsto \mathsf{leak/theft}\\
\mathsf{loss} &\mapsto \mathsf{loss/theft}.
\end{aligned}$$

Observe that the transform satisfies the two conditions related to credential knowledge laid out before.

Lemma 15. *If a mechanism is unsuccessful in a scenario σ, then it is also unsuccessful in a worse scenario $\widehat{\sigma}$. That is, for all mechanisms M, $Suc(M, \sigma) = \mathit{False} \implies Suc(M, \widehat{\sigma}) = \mathit{False}$.*

Note that proving the above also implies that success in a worse scenario $\widehat{\sigma}$ implies success in σ.

Proof. We prove by contradiction. That is, assume that the attacker succeeds in a scenario σ but the user succeeds in the transformed scenario $\widehat{\sigma}$.

Let the successful strategy for the user in $\widehat{\sigma}$ be S^U. Since $\widehat{C^U_\sigma} \subseteq C^U_\sigma$, the user can employ the strategy S^U in the scenario σ. By our initial assumption, the attacker succeeds in σ. Let the successful attacker strategy be S^A. S^A must win some executions against any user strategy including S^U. So we have $Win_\sigma(S^U, S^A) = \mathrm{A}$.

Since $C^A_\sigma \subseteq \widehat{C^A_\sigma}$, the attacker can employ the strategy S^A in the scenario $\widehat{\sigma}$. Any execution in the scenario $\widehat{\sigma}$ where the user employs S^U and the attacker employs S^A results in a win for the user. So we have $Win_{\widehat{\sigma}}(S^U, S^A) = \mathrm{U}$.

But σ and $\widehat{\sigma}$ use the same underlying mechanism M. And if the strategies of the two parties are same, then both $Win_\sigma(S^U, S^A) = \mathrm{A}$ and $Win_{\widehat{\sigma}}(S^U, S^A) = \mathrm{U}$ cannot be true due to Lemma 14 and we arrive at a contradiction. □

We now prove that $\mathcal{O}_3$ is made up of the priority and majority mechanisms only, i.e., Theorem 6.

Proof. Proving a complete maximal set requires satisfying three requirements per Definition 17:

1) All mechanisms in $\mathcal{O}_3$ must be maximal.
2) All mechanisms in $\mathcal{O}_3$ must be incomparable.
3) No other mechanism exists s.t. it is incomparable to each mechanism in $\mathcal{O}_3$.

The first requirement is met due to Lemma 9, Lemma 10 and Lemma 11. The second one is also met, as can be verified by inspecting the 76 mechanism profiles. (We cross-check this through code.)

We prove the third requirement using a similar approach taken in Theorem 5, but done at a larger scale. We use a constraint solver to show that no 3-credential mechanism M exists satisfying two types of constraints, explained below.

The first type of constraint encodes that M must be incomparable with any mechanism in $\mathcal{O}_3$. Satisfying it requires that for each mechanism $M' \in \mathcal{O}_3$, there exists at least one scenario $\sigma \in \mathit{prof}(M)$ that satisfies $\sigma \notin \mathit{prof}(M')$. This scenario σ must be a with-safe scenario due to Theorem 2. Rephrasing the above, we get $\exists \sigma \in \mathit{prof}(M)$ such that $\sigma \in \Sigma_{\mathsf{ws}} \setminus \mathit{prof}(M')$.

Observe that for any $M' \in \mathcal{O}_3$, $\mathit{prof}(M')$ contains all the with-safe scenarios except some with-safe-with-theft scenarios. Therefore $\Sigma_{\mathsf{ws}} \setminus \mathit{prof}(M') \subset \Sigma_{\mathsf{wswt}}$. And consequently, $\mathit{prof}(M)$ must include at least one with-safe-with-theft scenario.

So to prove that no mechanism M exists, it suffices to prove that $\mathit{prof}(M)$ does not contain a with-safe-with-theft scenario. Let $\mathit{prof}_{\mathsf{wswt}}(M)$ denote the subset of $\mathit{prof}(M)$ containing just the with-safe-with-theft scenarios. We want to show that $|\mathit{prof}_{\mathsf{wswt}}(M)| = 0$.

We prove this by consider the set of all subsets of the 18 with-safe-with-theft scenarios satisfying Theorem 1, i.e., no two scenarios in a subset are complements of each other. Denote this set by Δ ($|\Delta| = 3^9$ because the 18 with-safe-with-theft scenarios can be arranged into 9 rows with each row containing 2 complement scenarios. And we have three ways of making a selection in each row: {none, first scenario, second scenario}). Observe that this is the set of all possible values for $\mathit{prof}_{\mathsf{wswt}}$, i.e., for any mechanism M, it must be that $\mathit{prof}_{\mathsf{wswt}}(M) \in \Delta$.

And we want to find a set in Δ satisfying the two constraints. The first constraint, explained before, is $\forall M' \in \mathcal{O}_3, prof_{\mathsf{wswt}}(M) \cap (\Sigma_{\mathsf{ws}} \setminus prof(M')) \neq \phi$.

The second constraint relies on the following observation. If the mechanism wins in a with-safe-with-theft scenario σ, Theorem 1 implies that it fails in the complement scenario $\overline{\sigma}$. But then, due to Lemma 15, the mechanism fails in any scenario worse than $\overline{\sigma}$. For example, $prof(M)$ cannot contain the two scenarios $\sigma = \{\mathsf{theft}, \mathsf{theft}, \mathsf{safe}\}$ and $\sigma' = \{\mathsf{loss}, \mathsf{safe}, \mathsf{theft}\}$. This is because the mechanism fails in the complement of σ, $\overline{\sigma} = \{\mathsf{safe}, \mathsf{safe}, \mathsf{theft}\}$, and σ' is a worse scenario than $\overline{\sigma}$.

The second constraint is $\forall \sigma \in prof_{\mathsf{wswt}}(M), \nexists \sigma' \in prof_{\mathsf{wswt}}(M)$ s.t. the complement $\overline{\sigma}$ is worse than σ'.

The constraint solver outputs the empty set, i.e., no such $prof_{\mathsf{wswt}}(M)$ exists and hence $|prof_{\mathsf{wswt}}(M)| = 0$. This completes the proof that $\mathcal{O}_3$ is complete. □

Lemma 16. $|\mathcal{O}_3^{\mathsf{iso}}| = 14$.

Proof. Finding the number of non-isomorphic majority-based mechanisms requires some more work. It requires an alternative explanation for how to break ties. Breaking ties can be done in two ways: using *linear priority* (or) *cyclic priority*. Linear priority rules are same as before, i.e., we pick a permutation of the n-credentials and use it to break ties, e.g., $c_2 \sqsupset c_1 \sqsupset c_3$. On the other hand, cyclic priority rules involve picking a cyclic permutation, e.g.,$c_1 \sqsupset c_2$, $c_2 \sqsupset c_3$ and $c_3 \sqsupset c_1$.[1]

In total there are 6 different linear priority rules and 2 different cyclic priority rules. We can pick one of eight different rules to break ties between the 3 1-credential sets, and similarly for the 3 2-credential sets, leading to $8 * 8 = 64$ mechanisms.

Now the isomorphisms are immediate based on whether linear / cyclic rules are used for breaking ties among the 1-credential and 2-credential cases.

- Both linear: 36 mechanisms, 6 non-isomorphic.
- Both cyclic: 4 mechanisms, 2 non-isomorphic.
- Linear in one and cyclic in another: 24 mechanisms, 4 non-isomorphic.

□

Appendix B
Argent Extended Analysis

A. Two guardians

Note that due to Theorem 2, the mechanism fails in any scenario where all the credentials are unsafe. We discuss about other scenarios below.

If all the guardians are safe and c_o is unsafe (scenarios at the right bottom corners of **??**), then the mechanism is successful. The user's strategy is to change the owner and it is winning because canceling an owner change requires two credentials.

c_o *safe:* In case one of the guardians is leaked or theft, the mechanism fails. The attacker's winning strategy is similar to the one guardian case, namely, make the wallet unusable by bringing the automaton to the owner change state C_o (requires one guardian) perpetually.

In case one of the guardians is lost while the other is safe or if both guardians are lost, the mechanism succeeds. This is because the user can successfully change the guardian, i.e., both add a new guardian and revoke lost guardians, as changing guardian only requires the owner credential.

c_o *lost:* If one of the guardians is safe and the other is lost, then the mechanism wins because the user can successfully change the owner followed by changing the guardian.

If one guardian is safe and the other is leaked or theft, then the mechanism fails. In this case, the attacker's strategy is to try and change the owner credential. The user will be able to cancel it as it knows two credentials, but the attacker can initiate another owner change immediately, thereby creating an infinite loop.

c_o *leaked:* If one of the guardians is safe and the other is lost, then the mechanism wins. Observe that the user can successfully change the owner using the safe guardian (the attacker cannot cancel it). After an owner change, we move to a different scenario where c_o becomes safe. This case was analyzed before and the mechanism wins.

In case one of the guardians is leaked or theft, the mechanism fails. A winning strategy for the attacker is to move money to an attacker-controlled address, i.e., move the automaton to the final state F_0 (requires owner and a guardian).

c_o *theft:* If one of the guardians is leaked or theft, the mechanism fails because the attacker can move to the final state F_0 using owner and a guardian.

If one of the guardians is safe and the other is lost, then the mechanism succeeds. This is because the user can successfully execute an owner change, thereby making c_o safe and moving to a scenario where the mechanism wins.

[1]Note that we misuse the symbol "$\sqsupset$" a little since the relation is not transitive in a cyclic rule unlike the linear rule.

Appendix C
Priority Mechanism in Solidity

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract PriorityWallet {

    uint public numCredentials;
    mapping(address => uint) public priority;

    bool public withdrawalInProgress = false;
    uint public maxClaimID = 0;

    struct Details {
        bool[] supporters;
        uint amount;
        address payable addr;
    }
    Details[] public claimDetails;

    uint64 constant public delay = 30;
    uint64 public expiryTime;

```

```
23      // Set the guardians and the priority vector.
            Ideally, you'd also want to deposit some
            money.
24      constructor(address[] memory credentialList_)
            payable {
25          numCredentials = credentialList_.length;
26          for (uint i = 0; i < credentialList_.length;
                 i++) {
27              priority[credentialList_[i]] = i + 1;
28          }
29      }
30
31      function getBalance() public view returns (uint)
             {
32          return address(this).balance;
33      }
34
35      // start a new withdrawal. Internally creates a
            new claim
36      function initiateWithdrawal() public {
37          assert (priority[msg.sender] > 0);
38          assert (!withdrawalInProgress);
39
40          withdrawalInProgress = true;
41          expiryTime = uint64(block.timestamp + delay)
                ;
42          // purge previous claim data (if any)
43          delete claimDetails;
44          maxClaimID = 0;
45      }
46
47      function getClaimSupporters(uint claimID) public
             view returns (bool[] memory) {
48          return claimDetails[claimID].supporters;
49      }
50
51      // adds approval to an existing claim
52      function addApproval(uint claimID) public {
53          assert (withdrawalInProgress);
54          assert (claimID < maxClaimID);
55          assert (uint64(block.timestamp) <=
                expiryTime);
56          assert (priority[msg.sender] > 0);
57
58          claimDetails[claimID].supporters[priority[
                msg.sender]] = true;
59      }
60
61      function createNewClaimForWithdrawal(uint amount
            , address payable ToAddress) public returns
            (uint claimID) {
62          assert (priority[msg.sender] > 0);
63          assert (withdrawalInProgress); // otherwise
                call initiateWithdrawal
64          assert (uint64(block.timestamp) <=
                expiryTime);
65
66          bool[] memory supporters = new bool[](
                numCredentials + 1);
67          supporters[priority[msg.sender]] = true;
68          claimDetails.push(Details(supporters, amount
                , ToAddress));
69
70          claimID = maxClaimID; // create new claimID
71          maxClaimID = maxClaimID + 1;
72      }
73
74      function withdraw() public {
75          assert(uint64(block.timestamp) > expiryTime)
                ;
76          assert(withdrawalInProgress);
77
78          bool[] memory potentialWinners = new bool[](
                maxClaimID);
79          for (uint c = 0; c < maxClaimID; c++) {
80              potentialWinners[c] = true;
81          }
82
83          for (uint p = 1; p <= numCredentials; p++) {
84              bool foundAny = false;
85              for (uint c = 0; c < maxClaimID; c++) {
86                  if (potentialWinners[c] &&
                        claimDetails[c].supporters[p]) {
87                      foundAny = true;
88                  }
89              }
90
91              if (foundAny) {
92                  for (uint c = 0; c < maxClaimID; c
                        ++) {
93                      if (potentialWinners[c] && !
                            claimDetails[c].supporters[p
                            ]) {
94                          potentialWinners[c] = false;
95                      }
96                  }
97              }
98          }
99
100         for (uint c = 0; c < maxClaimID; c++) {
101             if (potentialWinners[c]) {
102                 uint winningClaimID = c;
103                 bool sent = claimDetails[
                        winningClaimID].addr.send(
                        claimDetails[winningClaimID].
                        amount);
104                 require(sent, "Failed to send Ether"
                        );
105                 withdrawalInProgress = false;
106                 break;
107             }
108         }
109     }
110 }
```